

TECHNISCHE UNIVERSITÄT DRESDEN

Fakultät Informatik  
Institut für Systemarchitektur  
Professur Rechnernetze

# Echtzeitvisualisierung für große Datenmengen

## Masterarbeit

zur Erlangung des akademischen Grades  
Master-Medieninformatiker

Robin Wieruch  
Geboren am 28. Mai 1988 in Berlin

Erstgutachter: Prof. Dr. rer. nat. habil. Dr. h. c. Alexander Schill  
Zweitgutachter: Dr.-Ing. Thomas Springer  
Betreuer: Dipl.-Inf. Robert Lübke  
Dipl.-Inf. Thomas Walther

Dresden, den 5. Oktober 2014



## **Erklärung**

Hiermit erkläre ich, Robin Wieruch, die vorliegende Masterarbeit zum Thema

### **Echtzeitvisualisierung für große Datenmengen**

selbstständig und ausschließlich unter Verwendung sowie korrekter Zitierung der im Literaturverzeichnis angegebenen Quellen und Hilfsmittel verfasst zu haben.

Robin Wieruch, Dresden, den 5. Oktober 2014



## AUFGABENSTELLUNG FÜR DIE MASTERARBEIT

Name, Vorname:	Wieruch, Robin		
Studiengang:	Med.-Inf. (MA)	Matrikelnummer:	3607287
Forschungsgebiet:	Pervasive Computing and Collaboration	Forschungsprojekt:	NESSEE
Betreuer:	Dipl. Inf. Robert Lübke Dipl. Inf. Thomas Walther	Externe(r) -	Betreuer:
Verantwortlicher Hochschullehrer:	<i>Prof. Dr. rer. nat. habil. Dr. h. c. Alexander Schill</i>		
Beginn am:	01.05.2014	Einzureichen am:	09.10.2014
Thema:	Echtzeitvisualisierung für große Datenmengen		

### ZIELSTELLUNG

In vielen großen verteilten Systemen fallen zur Laufzeit große Mengen an Daten an, z.B. Log-, Anwendungs- und Nutzerdaten. Diese Daten werden beispielsweise zur Analyse des Programmablaufs und zur Fehlersuche benötigt. Da die Daten allerdings auf vielen einzelnen Komponenten des Systems verteilt sind, ist deren komplette Auswertung oftmals sehr mühsam. Weiterhin sollen anfallende Daten zumeist bereits während der Ausführung eines Programmes ausgewertet werden. Durch die große Menge und Vielzahl verschiedener Daten ist eine entsprechende Visualisierung bei der Auswertung essentiell.

Das Ziel dieser Arbeit ist es, Konzepte zur Speicherung und Visualisierung großer Datenmengen in Echtzeit zu entwickeln. Dabei sollen die Speicher- und Visualisierungsdienste durch eine eigenständige Komponente angeboten werden. Die Speicherung soll effizient sein und Methoden der Filterung und Ausdünnung der Daten implementieren, falls eine bestimmte Speicherobergrenze erreicht ist. Die Visualisierung der Daten soll konfigurierbar sein und es sollen alle notwendigen Diagrammtypen angeboten werden. Die entwickelten Konzepte sollen im Rahmen der Test- und Emulationsplattform NESSEE implementiert werden. Insbesondere soll die Nutzung des Visualisierungsdienstes anhand von Live-Testdaten der Plattform beispielhaft implementiert werden. In der Evaluation ist die Erfüllung der Anforderungen, allen voran die Skalierbarkeit der eigenen Lösung, nachzuweisen.

### SCHWERPUNKTE

- Grundlagen und verwandte Arbeiten zur Echtzeitvisualisierung großer Datenmengen
- Analyse der Anforderungen an die eigenständige Speicher- und Visualisierungskomponente
- Entwurf von Konzepten zur effizienten Speicherung und Visualisierung von großen Echtzeitdaten
- Implementierung und Evaluierung der Konzepte im Rahmen der NESSEE Emulationsplattform

---

Verantwortlicher Hochschullehrer

*Prof. Dr. rer. nat. habil. Dr. h. c. Alexander Schill*



## Kurzzusammenfassung

Seit dem Aufkommen von *Big Data* werden vermehrt Analysen auf riesige Mengen von Daten durchgeführt. Daneben ermöglicht es die *Cloud* durch sogenanntes *Cloud-based Analytics* komplexe Algorithmen zur Analyse und Datenverarbeitung auszulagern. Weitergehend können Visualisierungen der Daten und Interaktionen auch für den Menschen zum besseren Verständnis und zur eigenständigen Exploration genutzt werden. Unternehmen erhoffen sich durch die Datenanalyse insgesamt eine Optimierung von Prozessen in vielfältigen Anwendungsgebieten. Dabei ist der Umgang mit großen Datenmengen und deren Echtzeitvisualisierung eine weitreichende Disziplin, welche unter anderem Forschungsgebiete der Visualisierung, Kommunikation, Datenverarbeitung und Speicherung vereint. Diese enden möglichst in einer ausbalancierten Komposition aus Komponenten für eine performante, interaktive und effiziente Echtzeitvisualisierung von großen Datenmengen. In dieser Arbeit werden die Forschungsgebiete im Hinblick auf die Umsetzung eines eigenen Services untersucht. Ein breites Spektrum an Grundlagen und verwandten Arbeiten ebnet den Weg zur Konzeption und Implementierung einer PIPELINR genannten Anwendung zur Echtzeitvisualisierung von großen Datenmengen.



# Inhaltsverzeichnis

<b>1. Einleitung</b>	<b>2</b>
<b>2. Grundlagen</b>	<b>4</b>
2.1. Datenverarbeitung . . . . .	4
2.1.1. Data-Mining . . . . .	5
2.1.2. Datenanalyse . . . . .	7
2.1.3. Datenreduktion . . . . .	10
2.2. Datenvisualisierung . . . . .	11
2.2.1. Datenstruktur . . . . .	12
2.2.2. Visualisierungsformen . . . . .	14
2.2.3. Interaktionstechniken . . . . .	20
2.3. NESSEE . . . . .	26
<b>3. Verwandte Arbeiten</b>	<b>30</b>
3.1. Verteilte Verarbeitung von Daten . . . . .	30
3.2. Fortgeschrittene Visualisierungsformen für zeitbasierte, multivariate Daten	37
3.3. Echtzeitvisualisierung und Änderungen des Kontextes . . . . .	39
3.4. Systeme zur Datenexploration von zeitbasierten Daten . . . . .	41
3.5. Fazit . . . . .	46
<b>4. Anforderungsanalyse</b>	<b>48</b>
4.1. Anwendungsfall: NESSEE . . . . .	48
4.2. Nicht-funktionale Anforderungen . . . . .	49
4.3. Funktionale Anforderungen . . . . .	50
<b>5. Konzepte zur eigenen Anwendung: Pipelinr</b>	<b>54</b>
5.1. Architektur . . . . .	54
5.2. Backend . . . . .	55
5.2.1. Datenstruktur . . . . .	55
5.2.2. Fassade aus Schnittstellen . . . . .	56
5.2.3. Module und Kernfunktionen . . . . .	57
5.2.4. Datenanalyse . . . . .	58
5.2.5. Erweiterte Kommunikation zum Frontend . . . . .	59
5.3. Frontend . . . . .	60
5.3.1. Visualisierung . . . . .	60
5.3.2. Interaktionsmöglichkeiten . . . . .	62

## *Inhaltsverzeichnis*

---

5.3.3. Verknüpfung mit Pipelinr Konsumenten . . . . .	66
<b>6. Implementierung . . . . .</b>	<b>68</b>
6.1. Backend . . . . .	68
6.1.1. Schnittstellen . . . . .	69
6.1.2. Modularität . . . . .	69
6.1.3. Prozess der Datenreduzierung . . . . .	70
6.1.4. Datenspeicherung . . . . .	71
6.1.5. Websockets . . . . .	71
6.2. Frontend . . . . .	72
6.2.1. Anwendungsbereiche . . . . .	72
6.2.2. Echtzeitvisualisierung . . . . .	76
<b>7. Evaluation . . . . .</b>	<b>78</b>
7.1. Evaluation zur Datenübertragung . . . . .	78
7.2. Evaluation zum Sampling . . . . .	81
7.3. Evaluation zur Visualisierung . . . . .	84
7.4. Evaluation der weiteren Anforderungen . . . . .	87
7.5. Fazit . . . . .	89
<b>8. Ausblick . . . . .</b>	<b>92</b>
<b>9. Zusammenfassung . . . . .</b>	<b>94</b>
<b>A. Anhang . . . . .</b>	<b>98</b>
A.1. Schnittstellenbeschreibung . . . . .	98
<b>Literatur . . . . .</b>	<b>100</b>

## Abbildungsverzeichnis

1.	Data-Warehouse im Kontext [Han, 2011] . . . . .	6
2.	Analyse zeitbasierter Daten . . . . .	7
3.	Klassifikation von Ausreißern . . . . .	8
4.	Visualisierung des Feldzugs von Napoleon gegen Russland (1812-1813) [Tufte, 1983] . . . . .	13
5.	Visualisierungsformen für zeitbasierte (univariate) Daten [Weber et al., 2001] . . . . .	15
6.	Line Graphs mit mehreren Zeitserien [Bostock, 2012] . . . . .	15
7.	Varianten von Stacked Graphs [Byron and Wattenberg, 2008] . . . . .	16
8.	Small Multiples [Heer et al., 2010] . . . . .	16
9.	Visualisierung vieler Zeitserien durch Horizon Graphs [Reijner and Software, 2008] . . . . .	17
10.	Aufbau eines Horizon Graphs in Anlehnung an [Reijner and Software, 2008]	18
11.	Horizon Graph mit Offset der negativen Werte [Heer et al., 2009] . . . . .	18
12.	Pixel-orientierte Visualisierung [Keim, 1996] . . . . .	19
13.	Ausschnitt der Tile Map von [Mintz et al., 1997] . . . . .	19
14.	Techniken zur interaktiven Verzerrung . . . . .	22
15.	Taxonomie der Interaktionsaufgaben von [Heer and Shneiderman, 2012] .	23
16.	Architektur von NESSEE [Lübke et al., 2013] . . . . .	26
17.	Einstellung von Netzwerkparametern in einem Test Case mittels TDL [Lübke et al., 2013] . . . . .	27
18.	Kafka als Nachrichtensystem [Jay Kreps, 2011] . . . . .	33
19.	Storm Topologie aus Spouts (Wasserhahn) und Bolts (Blitze) [Storm] .	34
20.	Apache Drill Architektur [Hausenblas and Nadeau, 2013] . . . . .	35
21.	Benchmark mathematischer Berechnungen im Vergleich zu C (C hat einen Wert von 1) [Bezanson et al., 2012] . . . . .	37
22.	Visualisierungsformen für zeitbasierte Daten . . . . .	38
23.	Ausschnitt eines Überblicks von Änderungen durch Echtzeitdaten in verschiedener Visualisierungsformen und der einhergehende Verlust des Kontextes [Krstajic and Keim, 2013] . . . . .	40
24.	medi+board [Kostkova et al., 2014] . . . . .	42
25.	Visual Analytics Tool von Keim et al. [Stoffel et al., 2013] . . . . .	44
26.	Line Graph Explorer [Kincaid and Lam, 2006] . . . . .	44
27.	Überblick der Architektur von PIPELINR . . . . .	55
28.	Datenstruktur einer Pipeline in PIPELINR . . . . .	56
29.	Kommunikationsablauf durch Publish-Subscribe Verfahren PIPELINR .	59

## Abbildungsverzeichnis

---

30.	Horizontale und vertikale Ausrichtung von Datensätzen einer Pipeline in der Hauptvisualisierung . . . . .	61
31.	Hauptvisualisierung von drei nominalen und zwei metrischen Datensätzen . . . . .	62
32.	Horizontale Ausrichtung der Hauptvisualisierung mit Analysedarstellung (A), Legende (B), Zooming-Leiste (C) und Tabelle nominaler Werte (D) . . . . .	63
33.	Das Markieren eines Teilbereichs im Overview-Bereich (blau) beeinflusst die anderen Visualisierungen. Die Detail-Bereiche werden auf den markierten Bereich hineingezoomt (grün). Weiterhin sind zwei nominale Werte in der Markierung betroffen, welche in der Liste aller nominalen Werte markiert werden (rot). Die Analysedarstellung passt ihre Berechnungen auf den eingeschränkten Bereich der Detail-Bereiche an (gelb). . . . .	65
34.	Überblick der Pipelines in Pipelinr . . . . .	73
35.	Abfrage einer Pipeline in Pipelinr . . . . .	74
36.	Dashboard in Pipelinr . . . . .	75
37.	Aufbau der Directives Struktur . . . . .	76
38.	Evaluation: Steigende Last durch kontinuierliche Schreiboperationen neuer Werte in einer Pipeline I . . . . .	79
39.	Evaluation: Steigende Last durch kontinuierliche Schreiboperationen neuer Werte in einer Pipeline II . . . . .	79
40.	Evaluation - Abfrage großer Pipelines bei variierender Größe . . . . .	80
41.	Evaluation - Vergleich der Performanz zweier Pipelines (mit und ohne Sampling) bei kontinuierlicher Last durch Schreiboperationen neuer Werte in einer Pipeline . . . . .	82
42.	Evaluation - Bearbeitungszeit des Samplings bei variierender Anzahl von Werten in der Pipeline . . . . .	83
43.	Evaluation - Maximale Speicherauslastungen (Anzahl der Werte) einer Pipeline bei periodischem Sampling . . . . .	83
44.	Evaluation - Darstellen einer Pipeline bei variierender Größe . . . . .	85
45.	Evaluation - Darstellen eines neuen Wertes bei variierender Größe . . . . .	86

## **Tabellenverzeichnis**

1.	Anforderungen an die Datenübertragung . . . . .	52
2.	Anforderungen an die Datenvisualisierung . . . . .	53
3.	Schnittstelle zum Backend für Pipelines, Datensätze und Werte . . . . .	69
4.	Übersicht der erfüllten Anforderungen . . . . .	90

---

## Abkürzungsverzeichnis

- SQL** Structured Query Language  
**RMI** Remote Method Invocation  
**CRUD** Create, Read, Update, Delete  
**npm** Node Packaged Modules  
**ORM** Object Relational Mapping  
**ODM** Object Data Mapping  
**REST** Representational State Transfer  
**API** application programming interface  
**SaaS** Software as a Service  
**KDD** Knowledge Discovery in Databases  
**URL** Uniform Resource Locator

## 1. Einleitung

Riesige Datenmengen fließen täglich durch das Internet und sammeln sich auf den Rechnern von Unternehmen und privaten Haushalten. Die Speicherkapazitäten und Bandbreiten wachsen kontinuierlich, wodurch das Volumen der gesammelten und beförderten Daten rasant steigt. Primär Unternehmen sammeln für ihre Zwecke große Datenmengen, wobei nur ein geringer Prozentsatz davon wirklich genutzt wird. Auch im Internet der Dinge werden durch Sensoren, eingebettete Rechner und *Wearables* viele, meist ungenutzte, Daten gesammelt. Darunter fallen auch zeitbasierte Daten [Golab and Özsü, 2003], wie etwa Temperaturmessungen aus dem *Smart Home*, Vitalwerte aus dem medizinischen oder *Wearable Computing* Bereich und Systemwerte aus dem Monitoring einer Serverfarm.

Im Gebiet des *Big Data* werden schon heute Petabytes von Daten durch Unternehmen verarbeitet und analysiert. Während für die Verarbeitung und Analyse der Daten verteilte Services bereitstehen, werden in der Datenanalyse vermehrt automatische Analysealgorithmen eingesetzt, um den großen Datenmengen, ohne Exploration der Daten durch einen Nutzer, entgegenzutreten. Unternehmen ziehen große Vorteile aus *Big Data Analytics*, da dadurch Bereiche des Kundensupports, Betrugserkenntnisse, Marketingstrategien und weiteren Optimierungen stetig verbessert werden können. Dennoch sind solche Systeme und Verfahren oftmals kostspielig und aufwändig für kleinere Unternehmen, weshalb es umso wichtiger ist, Datenanalyse der breiten Masse für die Informationsgewinnung zugänglich zu machen. [Leavitt, 2013]

Seit dem Einzug der Cloud werden Infrastruktur, Verarbeitungsleistung und Speicherkapazität auch für kleinere Unternehmen und Privatpersonen bezahlbar. Dadurch besteht die Möglichkeit, Ressourcen aus der Cloud für Analysebedürfnisse zu nutzen. Auch das *Cloud-based Analytics* erhält seit dem Aufkommen von *Big Data* immer mehr Aufmerksamkeit. Seitens Anbieter werden *Software as a Service (SaaS)* Lösungen zur Analyse von Daten angeboten. Darüber hinaus bieten Erweiterungen in Form von Web-Anwendungen eine Visualisierung der Daten und ihrer Analyseergebnisse an.

An diesem Punkt soll diese Masterarbeit zur Echtzeitvisualisierung großer Datenmengen anknüpfen. Die Intention ist es, einen durch Forschungsergebnisse gestützten Service zur Echtzeitvisualisierung großer Datenmengen zu konzipieren und implementieren, in dem es für Nutzer möglich ist, eigene Analysen durch verständliche, platzsparende und echtzeitfähige Visualisierungen sowie einfachen Interaktionsmöglichkeiten durchzuführen. Die Echtzeitvisualisierung großer Datenmengen zeigt viele Facetten auf. Es beinhaltet die ausbalancierte Komposition von Echtzeitfähigkeit durch Kommunikation, Visualisierung und Interaktion sowie Datenverarbeitung, deren Reduzierung und Speicherung. Die Innovation dieser Arbeit liegt in der Kombination der einzelnen Forschungsgebiete

---

und Technologien für die Echtzeitvisualisierung großer Datenmengen.

Zielführend dafür sollen die einzelnen Abschnitte zu den Grundlagen und verwandten Arbeiten eine Basis für die Konzeption der eigenen Anwendung schaffen. In den Grundlagen (Kapitel 2) wird dafür hauptsächlich auf die Themen Datenreduzierung, Datenanalyse, Visualisierung und Interaktion eingegangen. Daraufhin fassen die verwandten Arbeiten (Kapitel 3) von Methoden zur Datenverarbeitung über Visualisierungen bis hin zu gesamten System zur Analyse und Visualisierung von großen Datenmengen eine Vielzahl von wissenschaftlichen Arbeiten in diesen Bereichen zusammen. In der Anforderungsanalyse (Kapitel 4) werden die Ziele der zu konzipierenden und implementierenden Anwendung definiert. Im Anschluss gilt es, die Anwendung basierend auf den Anforderungen und unter zu Hilfenahme der Grundlagen und verwandten Arbeiten zu konzipieren (Kapitel 5) und implementieren (Kapitel 6). Eine abschließende Evaluation (Kapitel 7) gibt einen Überblick über die Performanz und erfüllten Anforderungen durch die implementierte Anwendung wieder. Der Ausblick bietet daraufhin Aufschluss über Erweiterungen der Anwendung basierend auf den einzelnen Forschungsgebieten und den implementierten Bereichen der Anwendung. Eine Zusammenfassung (Kapitel 9) verschafft schlussendlich einen Überblick dieser wissenschaftlichen Arbeit.

## 2. Grundlagen

Die Grundlagen gliedern sich in die zwei Bereiche Datenverarbeitung (Abschnitt 2.1) und Datenvisualisierung (Abschnitt 2.2).

Der erste Teil aus der Datenverarbeitung gibt einen zusammenfassenden Überblick über den Bereich des *Data-Minings* (Abschnitt 2.1.1). Hier wird der Begriff näher erläutert, Prozesse der Informationsgewinnung beschrieben und mögliche Datenbestände diskutiert. Weiterhin werden insbesondere für zeitbasierte Daten Analyseverfahren erläutert (Abschnitt 2.1.2) und Techniken zur Reduktion großer Datenmengen betrachtet (Abschnitt 2.1.3).

Der zweite Bereich gibt Aufschluss über Grundlagen zur Datenvisualisierung. Dazu werden die zu visualisierenden Datenstrukturen erläutert (Abschnitt 2.2.1), Visualisierungsformen für zeitbasierte Daten näher gebracht (Abschnitt 2.2.2) und mögliche Interaktionstechniken betrachtet (Abschnitt 2.2.3).

Abschließend stellt der letzte Bereich (Abschnitt 2.3) das Projekt NESSEE vor. NESSEE soll in dieser Arbeit als Anwendungsfall für die Echtzeitvisualisierung von großen Datenmengen dienen.

### 2.1. Datenverarbeitung

Täglich strömen Petabytes von Daten durch das World Wide Web und sammeln sich auf den Rechnern von Privatpersonen und Unternehmen. Dies ist das Resultat von stetig wachsenden Speicherkapazitäten und dem Begehr danach, immer mehr Informationen zu sammeln. Vor allem Unternehmen generieren für eigene Zwecke enorme Datensammlungen. Dennoch bleiben viele dieser Daten ungenutzt in Datenrepositories, sogenannten “*data tombs*”, welche nur selten besucht werden, verwahrt. Ein tatsächlicher Nutzen durch die Daten entsteht erst nach deren Analyse. Daher besteht immer mehr Bedarf an Anwendungen zur (automatischen) Analyse, um Daten in organisierte Informationen zu transformieren. Dieser Prozess wird auch *Data-Mining* genannt. Die Fülle an Daten und der Bedarf an Analyseanwendungen wird auch als “*data rich but information poor situation*” beschrieben. Durch diese Kluft werden heutzutage Entscheidungen nach wie vor oft auf Grundlage von Intuitionen getroffen als auf Basis des Datenbestands. [Han, 2011]

Die folgenden Abschnitte gliedern sich wie folgt: Im Abschnitt 2.1.1 wird auf den Begriff *Data-Mining* näher eingegangen, auf den Prozess der Informationsgewinnung und auf die verschiedenen Datenbestände. Abschnitt 2.1.2 geht insbesondere auf die Informationsgewinnung durch Analyse bei zeitbasierten Daten ein. Im Abschnitt 2.1.3 wird auf die Techniken der Datenreduktion bei großen zeitbasierten Datenmengen eingegangen.

## 2.1. Datenverarbeitung

---

### 2.1.1. Data-Mining

*Data-Mining* beschreibt das Extrahieren von Informationen aus großen Datenmengen. Der Begriff ist etwas unpassend gewählt, da aus den Daten die Informationen geborgen werden sollen und nicht die Daten. Daher wären Benennungen wie “*Knowledge-Mining*” oder “*knowledge mining from data*” für das semantische Verständnis besser geeignet. Herkömmlich geht es darum, aus großen Datenmengen Informationen zu gewinnen, welche manuell nicht mehr bearbeitet werden können. Jedoch finden jene Methoden auch Anwendung bei kleineren Datenbeständen, deren Ergebnisse von Nutzern eingesehen werden können. [Han, 2011]

Während der Begriff *Data-Mining* schnell für den gesamten Prozess der Informationsgewinnung verwendet wird, beschreibt es genau genommen nur die Analyse der Daten. Oftmals müssen die Daten vorher auf die Analyse vorbereitet werden. Dieser Prozess wird auch “*Knowledge Discovery in Databases (KDD)*” genannt. Für die Analysevorbereitung existieren die Teilprozesse *Clean*, *Integrate*, *Transformation*, *Reduction*. *Clean* bezeichnet das Säubern der Daten von Rauschen oder inkonsistenten Fragmenten. *Integrate* beschreibt das Zusammenführen mehrerer Datenquellen in einem Datenbestand. *Transformation* bedeutet, die Daten zu normalisieren, damit sie für Algorithmen effizienter genutzt werden können. Abschließend kann durch *Reduction* eine Datenreduzierung erfolgen, indem redundante Daten entfernt werden oder Daten durch *Clustering* zusammengeführt werden. [Han, 2011]

Beim Datenbestand kann es sich um unterschiedliche Datenrepositories handeln. Relationale Datenbanken oder *Data-Warehouses* sind zwei der möglichen Varianten, die unter anderem für die Datenanalyse genutzt werden.

**Relationale Datenbank** Eine relationale Datenbank basiert auf dem Konzept der Verknüpfung einer Sammlung von einzigartig benannten Tabellen, welche ein Set von Attributen (Spalten) beinhalten. Demnach wird ein Datenobjekt mit mehreren Attributen als Reihe in einer Tabelle dargestellt. Eine Spalte der Tabelle beinhaltet den einzigartigen Schlüssel der Objekte. Die Datenobjekte aus einer Tabelle werden durch die Schlüssel mit Datenobjekten anderer Tabellen verknüpft. In einem Bestellsystem können beispielsweise so Bestellungen und Rechnungen verschiedenen Kunden zugeordnet werden.

Operationen auf relationalen Datenbanken erlauben das Extrahieren der Informationen aus dem Datenbestand. Datenbanksprachen wie *Structured Query Language (SQL)* ermöglichen es so, beispielsweise alle Kunden mit einer bestimmten Anzahl an getätigten Bestellungen abzufragen. Eine weitere interessante Abfrage wäre es, alle Kunden zu ermitteln, welche eine bestimmte Menge an Bestellungen zurückgeschickt haben. So könnten Verkaufshäuser auf zu viel Reklamationen von Bestellungen reagieren. Auch

Preistrends von Artikeln lassen sich so beobachten. [Han, 2011]

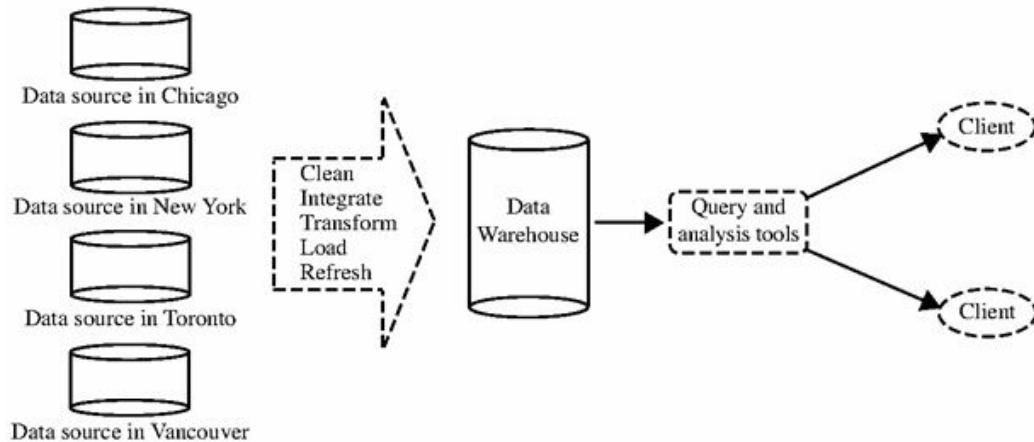


Abbildung 1: Data-Warehouse im Kontext [Han, 2011]

**Data-Warehouse** Das *Data-Warehouse* ist eine weitere Art der Datenhaltung, bei dem es sich um ein Repository für viele verstreute Datenbankbestände handelt. Unter einem einheitlichen Schema werden die Daten in einem *Data-Warehouse* zusammengeführt. Das Zusammenführen beinhaltet einige der bereits beschriebenen Schritte: *Clean*, *Integrate*, *Transform*, *Load* und *Refresh*. [Han, 2011] Abbildung 1 veranschaulicht diese Prozesse im Kontext des *Data-Warehouse*, indem global gestreute Datenquellen in einem *Data-Warehouse* zusammengeführt werden. Abschließend können Clients unter anderem Abfragen für Analysen an das *Data-Warehouse* stellen.

Die Daten werden aus einer historischen Sicht im *Data-Warehouse* zusammengeführt, wodurch beispielsweise Bestellungen der letzten drei Monate abgerufen werden können. Weiterhin werden die Daten zusammengefasst. Anstelle jede Bestellung einzeln abzuspeichern, werden pro Artikel die Anzahl der Bestellungen und weitere Werte kumuliert. [Han, 2011]

Neben den aufgezeigten *Data-Warehouses* und relationalen Datenbanken, lassen sich Datentypen unter anderem in zeitbasierte und räumliche Daten sowie die auf Multimedia (bspw. Audio, Video), Graphen und Netzwerk (bspw. Social Media) basierenden Daten differenzieren. [Han, 2011]

## 2.1. Datenverarbeitung

---

### 2.1.2. Datenanalyse

Im Hinblick auf zeitbasierte Daten können verschiedenen Informationen ermittelt werden. Durch bereits existierende Daten besteht die Möglichkeit, Vorhersagen durch sogenannte Trendanalysen zu treffen, welche dem Nutzer helfen, zukünftige Entscheidungen vorauszuplanen (Abbildung 2a). Auch charakteristische Entwicklungen der Daten während des Zeitablaufs, wie beispielsweise ein kontinuierlicher Anstieg, können durch auf Intervalle angewandte Durchschnittsberechnungen festgestellt werden (Abbildung 2b). Weiterhin besteht die Möglichkeit, Unregelmäßigkeiten in einer Zeitreihe zu entdecken (Abbildung 2c). Bei Hinzunahme weiterer zeitbasierter Daten können dann Schlussfolgerungen der Unregelmäßigkeit im Kontext aufgedeckt werden. Ferner besteht der Fall unregelmäßig vorkommende Daten im Datenstrom zusammenzufassen und Folgerungen daraus zu schließen.

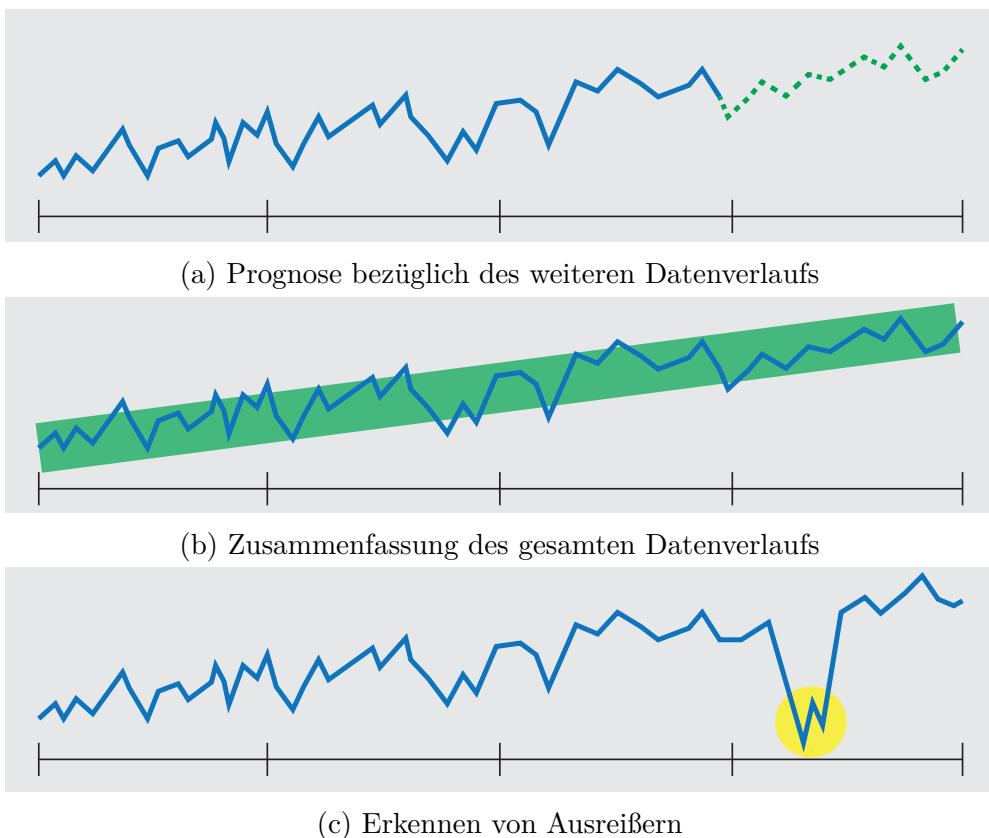


Abbildung 2: Analyse zeitbasierter Daten

Im Bereich des *Data-Minings* existieren verschiedene Techniken nach denen Muster aus Datenbeständen aufgedeckt werden. Die erbrachten Ergebnisse sind entweder prädiktiv oder deskriptiv. Während deskriptive Muster Eigenschaften des bestehenden Datensatzes beschreiben, charakterisieren prädiktive Muster durch den existierenden Datensatz mögliche zukünftige Eigenschaften. [Han, 2011] Im weiteren Verlauf sollen jeweils eine Technik zur prädiktiven und deskriptiven Mustererkennung erläutert werden. Die *Outlier Analysis* erkennt deskriptive Muster, indem *Outlier*, übersetzt als Ausreißer oder auch Anomalie, im Datenbestand aufgedeckt werden. Ausreißer werden in vielen Datenbeständen als unwichtiger Bestandteil vernommen. Bei zeitbasierten Daten können jedoch gerade diese Ausreißer interessanter sein als regulär vorkommende Werte. Daneben existiert zur Erkennung von prädiktiven Mustern die *Forecasting* Methode. Mit dieser ist es möglich, Trends einer zeitbasierten Datenreihe auszumachen. Auf beide Techniken wird im Weiteren eingegangen.

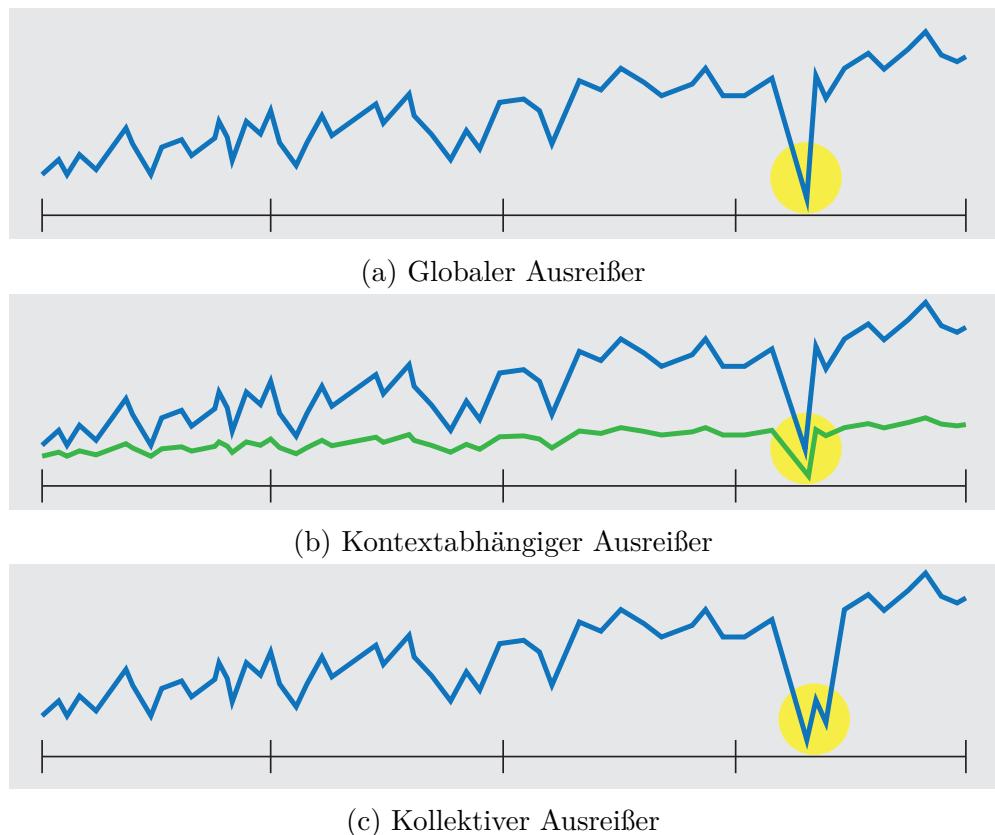


Abbildung 3: Klassifikation von Ausreißern

## 2.1. Datenverarbeitung

---

**Outlier Analysis** *Clustering* beschreibt das Zusammenführen von Daten auf Basis ähnlicher Attribute. Im Gegensatz zum *Clustering* werden bei der *Outlier Analysis* Daten mit ungewöhnlichen Attributen ausgemacht. Beide Techniken, das *Clustering* und die *Outlier Analysis*, stehen deshalb in enger Beziehung zueinander. Gewöhnlich sind Daten, die keinem *Cluster* hinzugefügt werden können, die sogenannten Ausreißer.

Ausreißer lassen sich in globale, kontextabhängige und kollektive Ausreißer klassifizieren. [Han, 2011] Globale Ausreißer (Abbildung 3a) weichen durch ihre Attribute wesentlich vom Rest des Datenbestands ab und stellen somit die einfachste Form der Ausreißer dar. Kontextabhängige Ausreißer (Abbildung 3b) müssen im Hinblick auf ihren Kontext untersucht werden. Wenn in einem System beispielsweise die CPU-Last für einen kurzen Moment steigt, könnte dies einen globalen Ausreißer ausmachen. Im Kontext betrachtet können mögliche Anwendungen ausgemacht werden, welche zu diesem Zeitpunkt mehr Leistung beansprucht haben und deshalb die CPU mehr ausgelastet wurde. Kollektive Ausreißer (Abbildung 3c) bestehen aus einer Gruppe von Ausreißern mit abweichenden Werten. Einzeln betrachtet müssen die Daten nicht zwingend Ausreißer darstellen. Abbildung 3 veranschaulicht noch einmal alle drei Typen.

Die Methoden zum Ausmachen von Ausreißern lassen sich in drei Arten klassifizieren: statistische, näherungsbasierte und *Clustering*-basierte Methoden. [Han, 2011] Die statistische Methode nimmt an, dass die Datenobjekte nach einem statistischen Modell verteilt sind. Alle Datenobjekte, die nicht in dem Modell liegen, sind Ausreißer. Die näherungsbasierte Methode wertet die Distanzen zum nächstgelegenen Datenobjekt aus. Ist eine Distanz wesentlich höher als der Durchschnitt der Distanzen, handelt es sich bei dem Datenobjekt um ein Ausreißer. Die *Clustering*-basierte Methode führt alle Datenobjekte zu *Cluster* zusammen. Einzelne Daten, die nicht in ein *Cluster* fallen, aber auch kleinere *Cluster*, werden als Ausreißer betrachtet.

**Forecasting** Ein wichtiger Bestandteil von Zeitreihen ist die Vorhersage von Werten. Dies hilft womöglich wichtige Entscheidungen für die Zukunft zu treffen. In der Literatur existieren viele Methoden zum Prognostizieren von Zeitwerten.

Autoregressive Techniken bemächtigen sich vorangehender Zeitwerte und Rauschtermen, um zukünftige Werte zu berechnen. Die Technik wurde 1927 von Yule [Yule, 1927] erstmals beschrieben und findet heutzutage in komplexerer Ausführung Anwendung in modernen Zeitreihenanalysen. Auch beim ARMA-Modell fließen frühere Werte und Rauschtermen mit ein. Das ARMA-Modell setzt sich aus dem AR-Modell (*auto regression*) und MA-Modell (*moving average*) zusammen. Beide Modelle errechnen ähnlich die zukünftigen Werte, dennoch “isoliert” das gleitende Mittel (*moving average*) im MA-Modell vorangehende Ausreißer durch eine Glättung der Datenwerte. [Stier, 2001]

### 2.1.3. Datenreduktion

Die automatische Analyse und Darstellung von großen Datenmengen fordert dem System viel Leistung ab. Während für die Analyse zahlreiche Daten ausgewertet werden müssen, bedarf es für die Darstellung der Visualisierung und für die Interaktionsmöglichkeiten viel Rechenzeit. Auch die Datenübertragung zu den jeweiligen Verantwortlichkeiten dauert länger, je größer die Datenmenge ist. Daher können Techniken der Datenreduktion angewandt werden, um den Datensatz zu verringern und dabei möglichst die Integrität des Datensatzes beizubehalten. Die anschließende Analyse der Daten sollte möglichst die selben Ergebnisse liefern, wie eine Analyse auf dem vollständigen Datensatz. Im Weiteren werden die Methoden *Dynamic Query Filter* und *Sampling* erläutert.

**Dynamic Query Filter** *Dynamic Query Filter* beschreibt das Abrufen eines gefilterten Teilbereichs des Datenbestands. [Ahlberg et al., 1992] Durch das Verschieben eines Schiebereglers [Eick, 1994], Anwählen oder Abwählen von Attributen oder das Festlegen eines Zeitbereichs können die Daten manuell durch den Nutzer gefiltert werden. In vielen vorangegangen Arbeiten wurde dadurch eine erhöhte Zufriedenheit beim Nutzer und eine verbesserte Performanz des Systems festgestellt. [Ahlberg et al., 1992, Williamson and Schneiderman, 1992] Vor allem bei zeitbasierten Daten bietet es sich an, den zu untersuchenden Zeitbereich vorher festzulegen.

**Sampling** *Sampling* bedeutet den Datenbestand nach einer ausgewählten Methode auszudünnen. In der Vergangenheit wurde oft daran gezweifelt, dass das Resultat der Analyse auf einen ausgedünnten Datenbestand das gleiche Resultat liefert, wie auf einem unveränderten Datenbestand. Statistiker haben beobachtet, dass “*a powerful computationally intense procedure operating on a subsample of the data may in fact provide superior accuracy than a less sophisticated one using the entire data base*” [Friedman]. Über die Zeit ergaben sich daher eine Vielzahl von Methoden zum Ausdünnen der Daten. Catlett untersuchte eine Reihe dieser Verfahren. [Catlett, 1991]

- *Random sampling* wählt zufällig eine Teilmenge an Daten aus.
- *Duplicate compaction* entfernt Duplikate aus dem Datensatz.
- *Stratified sampling* wird bei ungleichmäßig verteilten Daten angewendet. Bei einer höheren Dichte von Daten wird eher ausgedünnt als bei einer niedrigen Dichte. Die Verteilung soll am Ende möglichst gleichmäßig sein.

## 2.2. Datenvizualisierung

Das Darstellen und Erkunden von großen Datenmengen birgt trotz fortschrittlicher Hardware immer noch große Herausforderungen. Vor allem Visualisierungen von großen Datenmengen übersteigen oft das Fassungsvermögen der Nutzer. [Borgman, 1986] Überschaubare Visualisierungen und die Verwendung von Kodierungen wie Farbe, Größe und Form fördern das Verständnis beim Nutzer. [Bertin, 2011] Zunehmend unterstützen Interaktionsmöglichkeiten durch direkte Manipulation die Datenerkundung aber auch -eingrenzung. [Shneiderman, 1998]

Bei der Analyse von zeitbasierten Daten verfolgen Nutzer meist die Änderungen der Daten über die Zeit. Dafür werden Datenpunkte von verschiedenen Positionen verglichen. Im größeren Zusammenhang können dadurch Muster und Trends erkannt werden. [Aigner et al., 2007] Auch existieren weitere grundlegende Operationen beim Erkunden einer Visualisierung. [Andrienko and Andrienko, 2005, MacEachren, 2004]

- Auffinden von Datenobjekten
- Existenz eines Datenobjekts (mit zyklischem Verhalten) überprüfen
- Existenz von zusammenhängenden Datenobjekten ausmachen
- Frequenz der zeitlichen Änderungen feststellen
- Zeitliche Struktur der Daten eruieren
- Reihenfolge der Datenobjekte erkennen
- Ausmachen der Zeitspanne des Datensatzes

Visualisierungen lassen sich in zwei Teilgebiete gliedern: wissenschaftliche und Informationsvisualisierung. Die wissenschaftliche Visualisierung stellt ein Teilgebiet der Informationsvisualisierung dar. Hier werden vor allem physische Daten wie Volumen oder Strömungen dargestellt. Aref, Charles und Elvins beschreiben wissenschaftliche Visualisierung als "*when computer graphics is applied to scientific data for purposes of gaining insight, testing hypothesis, and general education*". [Card et al., 1999] Dagegen fasst Informationsvisualisierung die Darstellung abstrakterer Daten zusammen. Sie wird beschrieben als "*the use of computer-supported, interactive, and visual representations of abstract data to amplify cognition*". [Card et al., 1999] Generell wird die Informationsvisualisierung verwendet, um Nutzern einen besseren Einblick in eine Domäne zu gewähren. Immer mehr rücken diese Art der Visualisierungen vom Laborumfeld in die

Öffentlichkeit. [Plaisant, 2004] Dadurch ist die Informationsvisualisierung auch immer mehr im Internet anzutreffen.

Die folgenden Abschnitte gliedern sich wie folgt: Im Abschnitt 2.2.1 wird auf die zu visualisierenden Datenstrukturen eingegangen. Auch werden zeitbasierte Daten näher definiert. Abschnitt 2.2.2 geht auf verschiedene Visualisierungsformen für zeitbasierte Daten ein. Abschließend gibt Abschnitt 2.2.3 Einblicke in Interaktionstechniken für (zeitbasierte) Visualisierungen.

### 2.2.1. Datenstruktur

Die zu visualisierenden Datensätze bestehen aus Datenobjekten, welche verschiedene Entitäten wie Mitarbeiter, Kunden oder Transaktionen abbilden können. Weiterhin werden die Datenobjekte durch ein oder mehrere Attribute beschrieben. Attribute charakterisieren Objekte und können nominaler, binärer, ordinaler oder numerischer Art sein. [Han, 2011]

- Nominale Attribute sind Beschreibungen wie Name oder Status eines Datenobjekts. Sie unterliegen keiner Rangfolge und tragen in der Informatik auch die Bezeichnung *Enumeration*.
- Binäre Attribute haben zwei Ausprägungen: wahr und falsch oder 1 und 0. In der Informatik tragen sie die Bezeichnung *Boolean*.
- Ordinale Attribute sind Beschreibungen wie Noten im Schulsystem, die einer Reihenfolge unterliegen.
- Numerische Attribute sind quantitativ und demnach Zahlenwerte. In der Informatik werden sie als *Integer* repräsentiert.

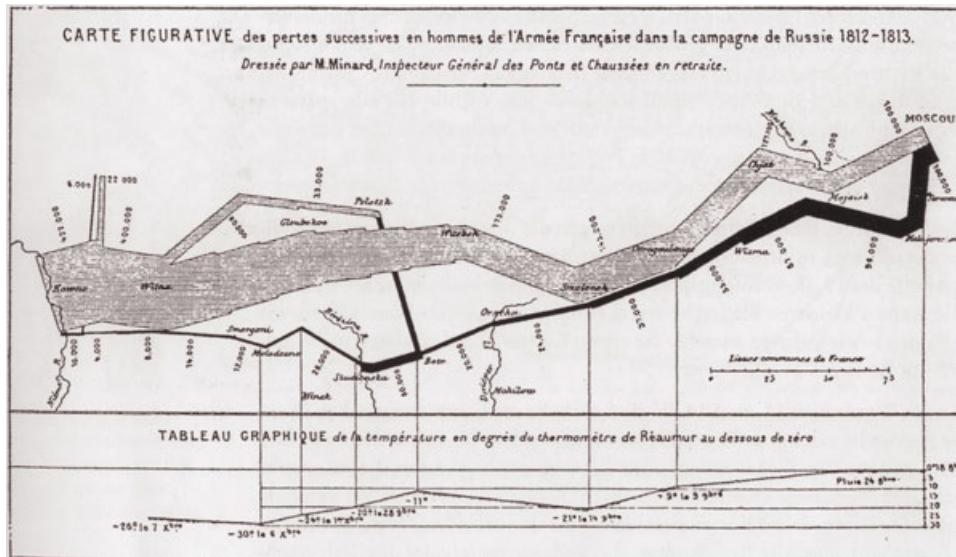
Numerische Attribute lassen sich weiterhin in die Teilateilattribute intervallskaliert und verhältnisskaliert zerlegen.

- Intervallskalierte Attribute haben keinen natürlichen absoluten Nullpunkt. Unter dieser Art von Attributen fallen beispielsweise ein Datum oder die Temperatur in Grad Celsius (Nullpunkt existiert, ist aber nicht natürlich sondern eher willkürlich).
- Verhältnisskalierte Attribute haben im Gegensatz zu intervallskalierten Attributen einen absoluten Nullpunkt. Unter dieser Art von Attributen fällt beispielsweise die Geschwindigkeit oder Körpergröße.

## 2.2. Datenvisualisierung

---

Wichtige Grundlagen zur Visualisierung dieser unterschiedlichen Attribute lieferte Jacques Bertin. Er differenzierte acht visuelle Variablen zur Kodierung von Attributen: Position, Größe, Form, Kontrast, Orientierung, Farbe, Textur und Bewegung. [Bertin, 2011]



Der Unterschied zwischen beiden Varianten liegt in der Validität. Zeitpunkte und ihre Werte gelten nur zum festgelegten Zeitpunkt. Die Werte zwischen den Zeitpunkten können interpoliert werden, sind aber eigentlich weitestgehend unbekannt.

Weiterhin lässt sich die Struktur der Zeit in drei Arten unterteilen: linear, zyklisch und verzweigt. [Frank, 1998]

- Lineare Strukturen sind geordnete Zeitwerte von Vergangenheit bis Zukunft.
- Zyklische Strukturen beschreiben beispielsweise ein Jahr oder eine Saison. Sie decken meist saisonale, wiederkehrende Verhalten auf.
- Verzweigte Strukturen werden als Graph dargestellt. Die verschiedenen möglichen Wege durch die Baumstruktur beschreibt mehrere zeitliche Szenarien.

### 2.2.2. Visualisierungsformen

Zeitbasierte Daten sind erfahrungsgemäß mit einem Zeitstempel oder Intervall versehen. Darüber hinaus können einzelne (univariate Daten) oder mehrere (multivariate Daten) Attribute dem Datum zugeordnet werden. Je nach Umfang und Art der Attribute, besteht die Möglichkeit, aus verschiedenen Visualisierungsformen zu wählen.

Univariate Daten lassen sich schlicht durch *Line Graphs* (Abbildung 5a) und *Point Graphs* visualisieren. *Line Graphs* sind als Erweiterung zu den *Point Graphs* anzusehen. Durch Interpolation der Daten werden die Zeitpunkte miteinander verbunden. Weiterhin bieten sich *Bar Graphs* für kumulative Daten an. *Spiral Graphs* [Weber et al., 2001] (Abbildung 5b) stellen zyklische Daten dar. Die Zeitachse wird hier durch eine Spirale repräsentiert. Daten können auf dieser Zeitachse als Punkte, Linie oder Balken dargestellt werden, wodurch wiederkehrende Muster in zyklischen Daten transparent aufgedeckt werden können (Vergleich Abbildung 5a und Abbildung 5b).

Die Visualisierung von zeitbasierten Daten deckt ein eigenes Kapitel in der Informationsvisualisierung. *Line Graphs* sind die einfachste Form zur Darstellung von zeitbasierten Daten. Durch die Zeitpunkt-Wert-Zuordnung werden Datenpunkte in einem Koordinatensystem eingezeichnet und mit (interpolierten) Linien miteinander verbunden. William Playfair veröffentlichte 1786 eine Reihe von Zeitreihenanalysen und legte damit einen wichtigen Grundstein für die Visualisierung von zeitbasierten Daten. [Playfair, 1786]

Heutzutage sind *Line Graphs* eine der meist genutzten Visualisierungsformen für statistische Daten. [Fitzpatrick, 1960] Trotzdem bestehen zeitbasierte Daten oft aus mehreren heterogenen Zeitserien [Hochheiser and Shneiderman, 2004], bei denen die Schwierigkeit darin besteht, diese Zeitreihen in einer möglichst platzsparenden Visualisierung unterzubringen. Dies bedeutet mehrere heterogene Zeitserien effizient in der Vertikalen auf

## 2.2. Datenvizualisierung

---

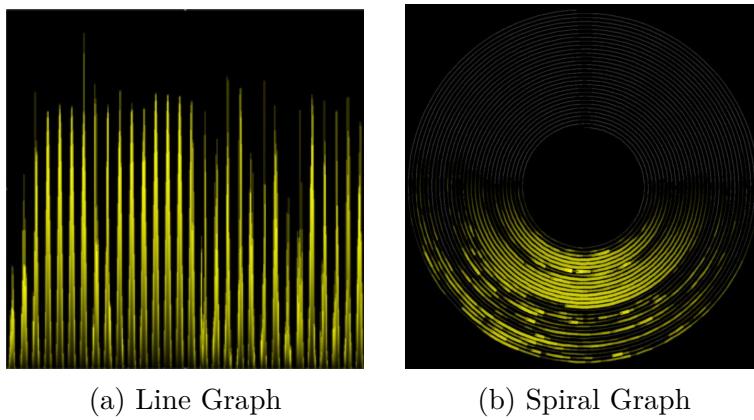


Abbildung 5: Visualisierungsformen für zeitbasierte (univariate) Daten [Weber et al., 2001]

einem Bildschirm darzustellen.

In kleineren Szenarien lassen sich unterschiedliche Zeitserien durch mehrere Y-Achsen (Abbildung 6a) oder mittels normalisierter Datenwerte in einer *Multi-Series Line Graph* darstellen (Abbildung 6b). Die Normalisierung der Werte hätte zur Folge, dass die Datenpunkte aus unterschiedlichen Zeitserien auf die gleiche Y-Achse abgebildet werden können.

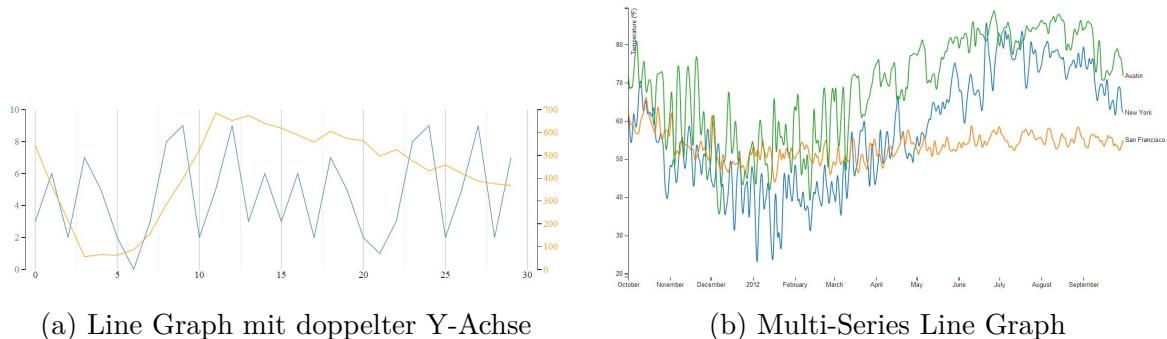


Abbildung 6: Line Graphs mit mehreren Zeitserien [Bostock, 2012]

In den Kriterien zur Evaluation von *Multiple Time Series* in der Arbeit von Javed et al. wird zwischen *shared space* und *split space* bei Zeitreihenvisualisierungen unterschieden. [Javed et al., 2010] In einem *shared space* teilen sich die Zeitserien ihre Umgebung (Abbildungen 6a, 6b). Dies birgt Vorteile für den Vergleich der Serien, sorgt jedoch bei mehreren Zeitserien für ein Durcheinander [Ellis and Dix, 2007] in der Visualisierung.

Im *split space* sind die einzelnen Serien deutlicher zu erkennen und können dadurch in einer größeren Menge dargestellt werden. Während unter *shared space* etwa Visualisierungen wie *Multi-Series Line Graphs* und *Stacked Graphs* gefasst werden, gehören *Horizon Graphs* und *Small Multiples* zu den *split space* Visualisierungen. Die benannten Visualisierungsformen werden im folgenden erläutert.

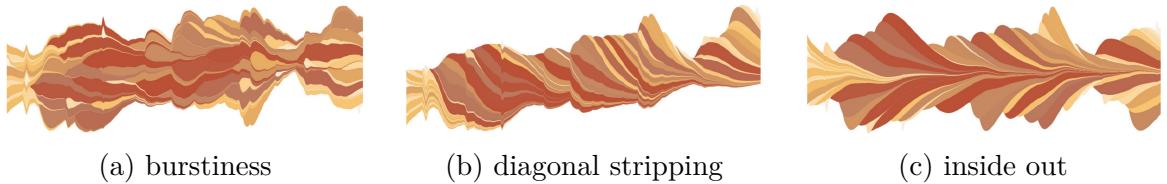


Abbildung 7: Varianten von Stacked Graphs [Byron and Wattenberg, 2008]

*Stacked Graphs* [Harris, 1999] eignen sich für mehrere übereinander gestapelte Zeitserien. Die Serien werden zur einprägsamen Unterscheidung eingefärbt und geben durch ihre Höhe den Wert zu einem bestimmten Zeitpunkt wieder. Da sie zu den *shared space* Visualisierungen gehören, müssen die Werte einem einheitlichen Attribut oder einer Normalisierung unterliegen. Byron und Wattenberg [Byron and Wattenberg, 2008] untersuchten später verschiedene Varianten von *Stacked Graphs* (Abbildung 7). So resultiert ein unsortierter Datensatz in einem *burstiness* Effekt (Abbildung 7a) und sortierte Datensätze in *diagonal stripping* und *inside out* Effekten (Abbildungen 7b, 7c).

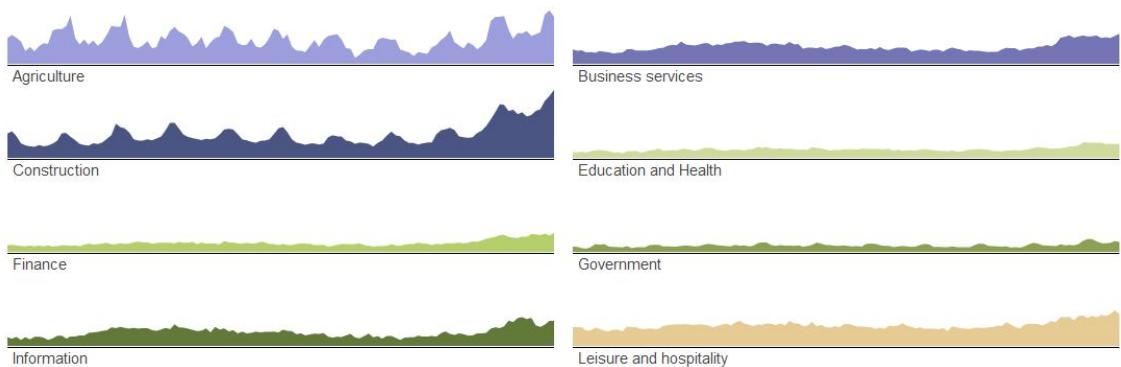


Abbildung 8: Small Multiples [Heer et al., 2010]

*Small Multiples* [Tufte, 1983] werden als eine Zusammenstellung von kleineren Visualisierungen gesehen. Tufte beschreibt die Technik eher als ein generelles Konzept für den Umgang mit mehreren homogenen Visualisierungen. Eine Herangehensweise wäre

## 2.2. Datenvizualisierung

---

es für jeden Zeitpunkt im Datensatz eine eigene Visualisierung zu verwenden. Für kumulierte Daten würde beispielsweise ein *Bar Graph* pro Zeitpunkt dargestellt werden. Bei Zeitreihen wird das Konzept zur Darstellung mehrerer Zeitserien durch *Line Graphs* verstanden (Abbildung 8). Durch die Trennung der Serien gehört diese Visualisierungsform zur *split space* Variante. Zur weiteren Unterscheidung können die einzelnen *Line Graphs* verschieden eingefärbt werden.

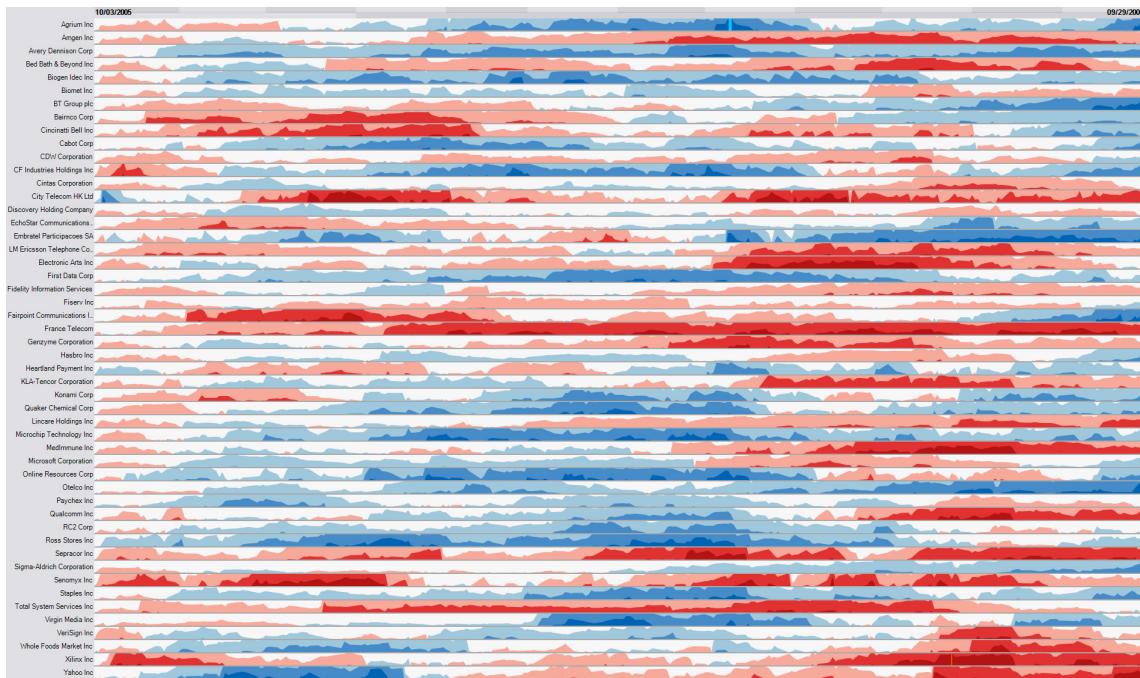


Abbildung 9: Visualisierung vieler Zeitserien durch Horizon Graphs [Reijner and Software, 2008]

*Horizon Graphs* [Reijner and Software, 2008] sollen den Vergleich vieler Zeitserien (Abbildung 9) ermöglichen. Die Visualisierungsform basiert auf der *Two-Tone-Pseudo Coloring* [Saito et al., 2005] Technik. Abbildung 10 illustriert die Gestaltung von einem *Line Graph* zu einem *Horizon Graph*. Der Hauptvorteil besteht in der Kompaktheit des Graphen. Nicht nur werden positive und negative Werte in einer Ebene dargestellt, auch die Werte selbst werden hier mittels "Bänder" gestaucht. Durch die dreifache Stauchung und der Spiegelung negativer Werte, wird nur noch ein Sechstel des Platzes benötigt. Die Bänderanzahl kann dabei variieren.

Heer et al. geben in [Heer et al., 2009] Vorschläge für die Anzahl der Bänder und ihrer optimalen Höhe. Nutzertests, unter anderem zur Genauigkeit und Fehlerfreiheit, erga-

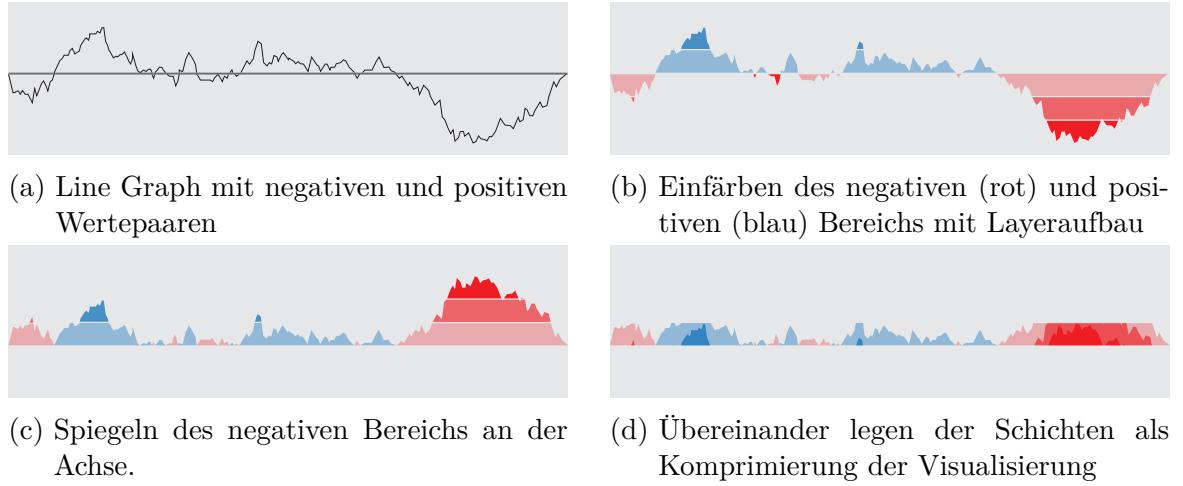


Abbildung 10: Aufbau eines Horizon Graphs in Anlehnung an [Reijner and Software, 2008]

ben für die Bänderanzahl, dass nicht mehr als drei Bänder genutzt werden sollten. Für die Höhe wurden nur ein und zwei Bänder bei einer Spiegelung der negativen Werte betrachtet. Optimal sind hier Höhen von 24 Pixel (ein Band) und 12 Pixel (zwei Bänder). Weiterhin führten Heer et al. neben den gespiegelten negativen Werten eine weitere Variante ein. Hier werden die negativen Werte mit einem *Offset* an den oberen Rand der Visualisierung verschoben (Abbildung 11). In der Evaluation brachte diese Variante keine Vorteile gegenüber den gespiegelten negativen Werten.

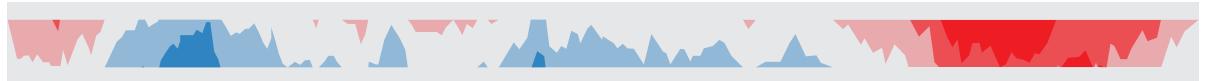
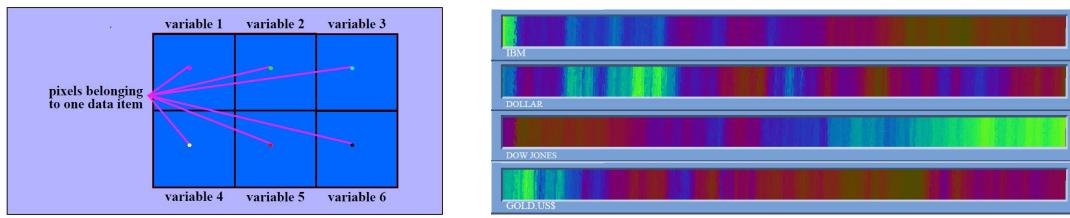


Abbildung 11: Horizon Graph mit Offset der negativen Werte [Heer et al., 2009]

Pixel-orientierte Visualisierungen [Keim, 1996] nutzen den maximal möglichen Darstellungsplatz, indem nur ein Pixel pro Datenwert für die Visualisierung genutzt wird. Die Einfärbung des Pixels markiert den Wert des Datums. Bei multivariaten Daten werden genauso viele Pixel wie Werte des Datums mit bestimmten Anordnungen nebeneinander dargestellt (Abbildung 12a). Beispielsweise werden für ein horizontales Arrangement die multivariaten Daten untereinander (vertikal) dargestellt und die sich so bildenden Liniensubblöcke in die horizontale Richtung aneinander gereiht (Abbildung 12b). Für univariete Daten ist es ausreichend ein Pixel oder auch Linienblock pro Datum darzustellen.

## 2.2. Datenvizualisierung

---



(a) Anordnung eines multivariaten Datums mit sechs Werten

(b) Horizontales Arrangement multivariater Daten

Abbildung 12: Pixel-orientierte Visualisierung [Keim, 1996]

Das *Recursive Pattern* [Keim et al., 1995] umfasst eine Teilmenge der Pixel-orientierten Visualisierungen. Diese Muster eignen sich vor allem für natürlich geordnete Datensätze. Zeitbasierte Daten wären hierbei ein Anwendungsfall, in dem die Daten meist ein horizontales oder vertikales Arrangement haben. Kombinationen aus beiden Arrangements sind auch möglich. Angenommen ein zeitbasierter Datensatz erstreckt sich über mehrere Monate: Hierbei könnte ein horizontales Arrangement einen Tag darstellen - die Tage werden in ihrer natürlichen Ordnung vertikal arrangiert. Abschließend formen abgegrenzte Blöcke die Monate des Datensatzes. Ähnlich wurde dieses Arrangement in der *Tile Map* von David Mintz et al. [Mintz et al., 1997] zur Darstellung der Luftqualität umgesetzt (Abbildung 13).

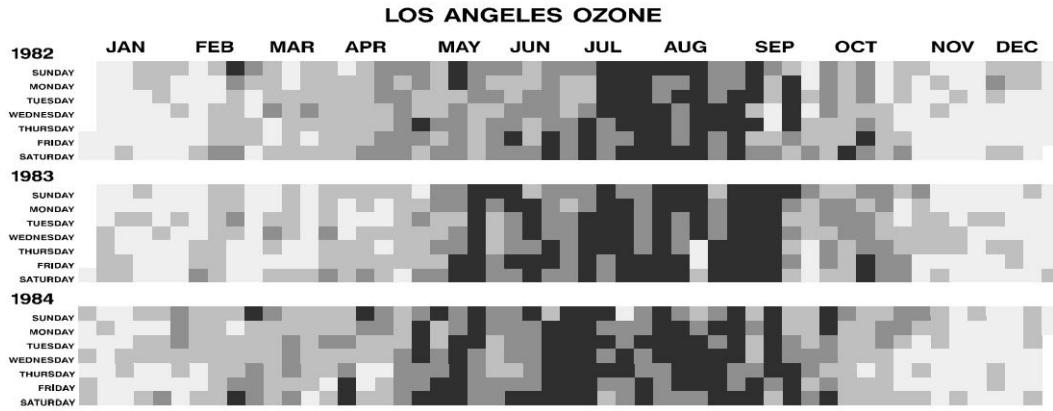


Abbildung 13: Ausschnitt der Tile Map von [Mintz et al., 1997]

### 2.2.3. Interaktionstechniken

Neben dem *Data-Mining* ist das eigenständige Erkunden der Daten durch den Nutzer ein wichtiger Analyseprozess. Es vereint die kognitiven Fähigkeiten des Menschen mit der Berechnungsstärke und Kapazität des Computers. Eine Visualisierung kann durch das Eingreifen des Nutzers dynamisch verändert werden. Die Interaktion mit den Daten und der Visualisierung ist daher ein wichtiger Prozess zum Auffinden von Informationen, was das Verständnis beim Nutzer fördert. [Cockburn et al., 2009] Es wird auch visuelles *Data-Mining* genannt.

Nach dem Mantra der visuellen Informationssuche von Shneiderman [Shneiderman, 1996] gibt es sieben Interaktionsaufgaben für Visualisierungen zu unterscheiden: *Overview*, *Zoom*, *Filter*, *Details-on-demand*, *Relate*, *History* und *Extract*. Während *Overview* das Gewinnen eines Überblicks über den gesamten Datenbestand beschreibt, können durch einen *Zoom* interessante Teilmengen von Datenobjekten betrachtet werden. Beim *Filter* wird eine Untergruppe von Datenobjekten anhand gefilterter Attribute dargestellt. Die *Details-on-demand* Interaktionsform ermöglicht es, für einzelne oder eine Gruppe von Datenobjekten mehr Details abzufragen. Die weiteren Interaktionsformen *Relate*, *History* und *Extract* beziehen sich auf das Vergleichen von Datenobjekten und ihren Werten, das Aufzeichnen von Aktionen und der Möglichkeit diese rückgängig zu machen sowie das Extrahieren einer Untergruppe von Datenobjekten.

Daniel A. Keim teilt den Interaktionsprozess in die Teilprozesse *Interaction* und *Distortion* auf. [Keim, 2002] Diese Teilprozesse sollen im Weiteren als dynamische Projektion, interaktives Filtern, *Zooming* und *Panning* [Van Wijk and Nuij, 2003], Verzerrung sowie *Linking* und *Brushing* zusammengefasst werden.

**Dynamische Projektion** Dynamische Projektionen helfen dabei multivariate Daten in einer zweidimensionalen Ebene darzustellen. Für jedes Paar von Attributen wird ein Koordinatensystem aufgespannt. Die Koordinatensysteme wachsen exponentiell mit der Anzahl der darzustellenden Attribute. Die Reihenfolge der dargestellten Koordinatensysteme kann zufällig, manuell oder nach Reihenfolge der Attribute angelegt werden.

**Interaktives Filtern** Interaktives Filtern wird benutzt, um Daten beispielsweise in einer Zeitspanne (*browsing*) oder mit Hilfe ihrer Attribute einzuschränken (*querying*). Während *browsing* einen zeitlichen Ausschnitt des Datenbestands liefert, selektiert *querying* nur Daten mit bestimmten Attributen. Während es sich hierbei um ein visuelles *Data-Mining* handelt, stehen im Hintergrund Methoden der Datenreduzierung wie das *Dynamic Query Filter* (Abschnitt 2.1.3). Das interaktive Filtern gehört zur Interaktionsaufgabe des Filterns aus den Interaktionsaufgaben des Mantras der visuellen Informationssuche von

## 2.2. Datenvisualisierung

---

Shneiderman.

**Interaktives Zooming und Panning** Vor allem bei großen Datenbeständen ist es wichtig, die Daten komprimiert in einer übersichtlichen Darstellung zu visualisieren. Das interaktive *Zooming* erlaubt es dem Nutzer, in Ausschnitte des Datenbestands einzutauchen ohne dabei den Überblick des gesamten Datenbestands zu verlieren. *Zooming* bedeutet nicht nur die Datenobjekte größer darzustellen, sondern auch die Detailstufen zu erhöhen. Dies können Pixel bei großer Zoomstufe, Icons bei mittlerer Zoomstufe und Beschriftungen bei niedriger Zoomstufe sein. Weiterhin besteht die Möglichkeit, den Datenbestand dynamisch anzupassen. Während bei großer Zoomstufe der Datenbestand beispielsweise durch *Sampling* ausgedünnt ist, werden beim Heranzoomen weitere Daten dynamisch nachgeladen. Die *Calendar View* [van Wijk and Van Selow, 1999] (Abbildung 22d) stellt eine beispielhafte Visualisierungsform für das *Zooming* dar.

Diese Art der Interaktion gehört zur Interaktionsaufgabe *Overview + Detail* aus Shneidermans Mantra der visuellen Informationssuche. Üblicherweise wird die Visualisierung in die zwei Teilbereiche *Overview* und *Detail* aufgeteilt. Während sich der *Detail*-Bereich stetig durch Interaktion ändert, zeigt der *Overview*-Bereich dauerhaft den gesamten Datenbestand. Allein eine Markierung im *Overview*-Bereich zeigt den aktuell visualisierten Ausschnitt aus dem *Detail*-Bereich.

Weiterhin existiert der Begriff des *Pannings*. Dies beschreibt die translative Navigation in der Visualisierung nach dem Hineinzoomen. Das *Panning* im *Detail*-Bereich wirkt sich sofort auf die Markierung im *Overview*-Bereich aus. Je weiter in die Ansicht hineingezoomt wurde, desto größer ist der mit *Panning* zurückzulegende Weg.

**Interaktive Verzerrung** Die interaktive Verzerrung zeigt dem Nutzer einen Teilbereich der Visualisierung in einem hohen Detaillierungsgrad, während ein Überblick der gesamten Daten durch einen Außenbereich mit niedrigen Detaillierungsgrad erhalten bleibt. Es existieren unterschiedliche Verzerrungstechniken. Eine Auswahl stellen Fisheye View [Sarkar and Brown, 1994] (Abbildung 14a), Perspective Wall [Mackinlay et al., 1991] (Abbildung 14b) und Bifocal Display [Leung and Apperley, 1994] (Abbildung 14c) dar. Diese Art der Interaktion gehört zur Interaktionsaufgabe *Overview + Detail* aus Shneidermans Mantra der visuellen Informationssuche.

**Interaktives Linking und Brushing** Das interaktive *Linking* und *Brushing* wird oftmals verwendet, um Datenobjekte miteinander zu verknüpfen. Bei dynamischen Projektionen besteht die Möglichkeit, einzelne Datenobjekte aus unterschiedlichen Koordinatensystemen zu verbinden (*Linking*). Durch eine Markierung eines Datenobjekts in

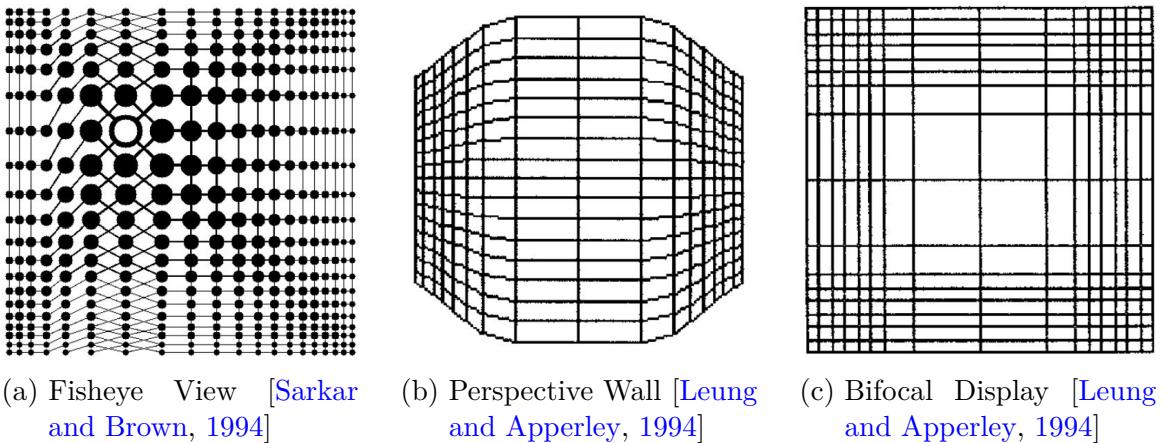


Abbildung 14: Techniken zur interaktiven Verzerrung

einem Koordinatensystem werden alle anderen zugehörigen Datenobjekte in den weiteren Koordinatensystemen hervorgehoben (*Brushing*). Diese Interaktion findet weiterhin Verwendung bei der Verknüpfung von Datenobjekten über unterschiedliche Visualisierungsformen hinweg. Weiterhin wird für große Datenbestände so eine verbesserte Übersicht bewahrt. Auch interaktive Änderungen, wie das *Zooming* in einem Bereich, kann andere Bereiche der Visualisierung betreffen (*Overview + Detail*).

Weitere Interaktionsmöglichkeiten werden in *Visual Analytics* beschrieben. Dabei handelt es sich um ein Forschungsgebiet zur Analyse von großen, oft komplexen, Datenmengen zur Exploration von Clustern, Trends, etc. Auch hier verschmelzen automatische und rechenintensive Analysemethoden der Maschinen mit den kognitiven Fähigkeiten des Menschen. Es beschreibt eine Zusammenarbeit zwischen Datenverarbeitungen und Visualisierungen sowie Interaktionstechniken. Die visuelle Analyse wird oftmals durch *Visual Analytics* Applikationen gestützt. In diesen Anwendungen ist es möglich, variable Datensätze unterschiedlichen Visualisierungsformen zuzuordnen. Üblicherweise bieten *Visual Analytics* Anwendungen eine Vielzahl von Visualisierungen an. Nutzer entscheiden selbst, welche Darstellung passend zur Exploration der Daten ist.

Im Bereich der *Visual Analytics* zählt schon das Visualisieren des Datensatzes zu den Interaktionstechniken. Eine Taxonomie zu Interaktionen von [Heer and Shneiderman, 2012] im Bereich *Visual Analytics* beschreibt 12 Aufgabentypen und teilt diese in 3 Kategorien ein (Abbildung 15).

Im Folgenden werden die einzelnen Interaktionsaufgaben aus [Heer and Shneiderman, 2012] kurz zusammengefasst. Diese können nach Heer und Shneiderman als Checkliste

## 2.2. Datenvizualisierung

---

<b>Data &amp; View Specification</b>	<b>Visualize</b> data by choosing visual encodings. <b>Filter</b> out data to focus on relevant items. <b>Sort</b> items to expose patterns. <b>Derive</b> values or models from source data.
<b>View Manipulation</b>	<b>Select</b> items to highlight, filter, or manipulate them. <b>Navigate</b> to examine high-level patterns and low-level detail. <b>Coordinate</b> views for linked, multi-dimensional exploration. <b>Organize</b> multiple windows and workspaces.
<b>Process &amp; Provenance</b>	<b>Record</b> analysis histories for revisit, review and sharing. <b>Annotate</b> patterns to document findings. <b>Share</b> views and annotations to enable collaboration. <b>Guide</b> users through analysis tasks or stories.

Abbildung 15: Taxonomie der Interaktionsaufgaben von [Heer and Shneiderman, 2012]

für die Entwicklung von Visualisierungen oder *Visual Analytics* Anwendungen betrachtet werden.

**Visualize** Normalerweise ist die Visualisierung historisch betrachtet im Zusammenhang mit einer eigenen Programmierung verbunden. In *Visual Analytics* Anwendungen sind viele Visualisierungsformen zur Nutzung bereits implementiert. Nutzer können aus einer Palette von Visualisierungen wählen, Daten einfügen und mit Achsen verknüpfen sowie Einstellungen wie Größe und Farbe festlegen.

**Filter** Nutzer von *Visual Analytics* Anwendungen visualisieren oftmals nur Teildatensätze zur Exploration von einzelnen Analysebereichen. Je nach Dimension und Art des Teildatensatzes resultiert dies häufig in verschiedenen Visualisierungsformen zur Erkennung unterschiedlicher Muster. Auch der Vergleich mehrerer Teildatensätze ist ein wichtiger Aspekt des Filters. Filtermethoden werden oft durch *Dynamic Query Filter* mittels Schieberegler, Radiobuttons, Listen oder Suchfelder ausgeführt.

**Sort** Eine Sortierung von Daten ermöglicht einen differenzierten Blick auf einen Datensatz und zeigt *Cluster* oder Trends von Datenpunkten. Oftmals werden bestimmte Sortierungen wie das “Sortieren nach Monaten” oder “absteigende Sortierung” verwendet.

**Derive** Die Herleitung von neuen Daten ist für die Analyse unumgänglich. Aus Werten von Datensätzen lassen sich Summen oder Durchschnittswerte für eine statistische

Auswertung bilden. Auch können neue Werte aus zwei bereits bestehenden Werten gebildet werden. Oftmals werden die neu hergeleitete Daten für weitere iterative Analysen verwendet.

Die bisher vorgestellten Interaktionsaufgaben ergeben die Möglichkeit, Visualisierungen durch eine *Data and View Specification* entstehen zu lassen. Die weiteren Interaktionsaufgaben sollen eine Manipulation der Visualisierung ermöglichen.

**Select** Das Selektieren von Datenbereichen ermöglicht die weitere Verarbeitung dieser Teildaten. Dies betrifft u.a. den markierten Bereich mittels Operationen beispielsweise zu Filtern oder neue Daten herzuleiten. Weiterhin kann der markierte Bereich auch in anderen Visualisierungen für den gleichen Datensatz hervorgehoben werden. In der wissenschaftlichen Arbeit [Holz and Feiner, 2009] können Bereiche einer Zeitserie markiert werden, um automatisch Muster in der gezeichneten Linie zu erkennen.

**Navigate** Die Darstellung von großen Datenmengen erfordert oftmals Navigationstechniken zur Betrachtung von Teilmengen des Datensatzes. Oft verwendete Techniken sind das *Zooming*, *Panning* und *Scrolling*. Im weitesten Sinne können auch *Focus + Context* Methoden wie die *Fisheye View* oder *Perspective Wall* als Navigationstechniken betrachtet werden.

**Coordinate** Das Koordinieren mehrere homogener oder heterogener Visualisierungsformen für den gleichen Datensatz ermöglicht in *Visual Analytics* Anwendungen eine differenzierte Sicht auf den Datensatz. Das Selektieren eines Teilbereichs hebt die Datenpunkte in allen Visualisierungen hervor (*Brushing* und *Linking*). Die Teilbereiche wiederum ergeben erneut eine neue Perspektive zur Exploration möglicher Muster.

**Organize** Das eigenständige Anordnen von einzelnen Visualisierungen und Bereichen ermöglicht Analysten die Erstellung eines personalisierten Dashboards. Eine mögliche Aufteilung wäre es, die Hauptvisualisierung mittig zum Rest zu positionieren. Außen herum lassen sich Legenden, Abfragemechanismen wie Suchfelder und Schieberegler sowie weitere einzelne Visualisierungen anordnen. Abschließend sollten Möglichkeiten zum Schließen, Minimieren, Maximieren, Öffnen und Erstellen von einzelnen Bereichen bereitgestellt werden.

Nach dem Erstellen der Visualisierung und der Bereitstellung von Interaktionstechniken folgt der iterative Analyseprozess. Die letzten vier Techniken sollen Analysten in *Visual*

## 2.2. Datenvisualisierung

---

*Analytics* Anwendungen dabei unterstützen.

**Record** Iterative Analyseprozesse stellen sich meist aus mehreren Hypothesen und Ergebnissen heraus. Sinnvoll ist es, diese Zwischenschritte zu dokumentieren und *Undo* sowie *Redo* Funktionen zur Verfügung zu stellen. Die Dokumentation kann Maus- und Tastatureingaben oder höhere semantische Inhalte, wie “Datensatz gefiltert” oder “Zooming um 50%”, beinhalten. Dem Aufzeichnen semantischer Inhalte wird eine komplexere Umsetzung nachgesagt, diese sind aber im Nachhinein eine wertvolle Funktion für Analysten. Die Dokumentation kann in einer *Timeline* dargestellt werden.

**Annotate** Die Kommunikation zwischen Analysten ist ein wichtiger Bestandteil in der Analyse von großen Datenmengen. Hierfür ist es notwendig, Punkte oder Bereiche mit Anmerkungen zu versehen. Letzteres ist ein mächtiges Werkzeug, denn hiermit können hilfreiche Einzeichnungen, wie etwa Pfeile, vorgenommen werden. Darüber hinaus machen es *Data-aware Annotations* möglich, Datenpunkte, die in mehreren Visualisierungen dargestellt werden, durch eine Anmerkung automatisch in allen Darstellungen zu annotieren.

**Share** Weiterhin werden die Kommunikation und der iterative Analyseprozess durch das Teilen von Ausschnitten der Visualisierung oder historischen Analyseschritten verstärkt. Demnach besteht die Möglichkeit, Prozesse einer Analyse auf mehrere Analysten aufzuteilen. Die *Visual Analytics* Anwendungen können dafür Bilder oder Teildatensätze exportieren. Auf höherem Niveau besteht auch die Möglichkeit, Zwischenschritte der Analyse auf eine [URL](#) abzubilden, die für eine spätere Wiederaufnahme oder Besichtigung des Analyseschritts aufgerufen werden kann.

**Guide** Als letzte Interaktionsaufgabe können in Visualisierungen Guides erstellt werden. Zielgruppe dieser Guides sind Laien oder Experten. Während Laien oftmals eine Einführung in die Visualisierung benötigen, bekommen Experten durch Guides wichtige Einblicke in einzelne Analyseschritte eines anderen Analysten. Heutzutage verwenden Zeitungsartikel oftmals das *Story Telling*, um Leser durch eine Infografik zu führen. Dies kann auf einfache Art und Weise durch Annotationen geschehen.

Abschließend betrachtet existieren unter dem Mantra der visuellen Informationssuche viele feingliedrige Interaktionstechniken, welche in diesem Abschnitt beschrieben wurden.

## 2.3. NESSEE

Im letzten Abschnitt zum Kapitel der Grundlagen wird das Projekt NESSEE (Network Endpoint Server Scenario Emulation Environment) vorgestellt. Darin wird deutlich, weshalb NESSEE für das in dieser Arbeit zu entwickelnde System zur Echtzeitvisualisierung von großen Datenmengen als Anwendungsfall dienen kann.

Das NESSEE Projekt ist eine Plattform zur Emulation von Netzwerk- und Anwendungsverhalten in verteilten Systemen. Es ermöglicht das Testen einer hierarchisch gestalteten Infrastruktur von verteilten Systemen durch Manipulation präziser Netzwerkparameter. Dadurch können bei der Netzwerkplanung Änderungen in der Infrastruktur vorzeitig auf Konsequenzen evaluiert werden. Oftmals werden solche Tests unter perfekten Laborbedingungen durchgeführt. In der realen Welt überqueren Datenpakete große Entfernung. Dabei gehen Pakete verloren, verzögern sich oder werden beschädigt. NESSEE emuliert dies durch klassische Netzwerkparameter in der Netzwerkschicht und höheren Schichten. [Lübke et al., 2013]

Die entscheidenden Parameter zur Emulation des Netzwerks sind Bandbreite, Verzögerung und Paketverlust. Auch weitere Parameter wie die Duplikation, Jitter und veränderte Reihenfolge der Pakete wurden im NESSEE Projekt bedacht. Weiterhin sind mobile Geräte, welche über andere Technologien kommunizieren (3G, 4G, WiFi), durch Parameter wie Übergabe zwischen Funkzellen und Verbindungsabbrüche emulierbar. [Lübke et al., 2014]

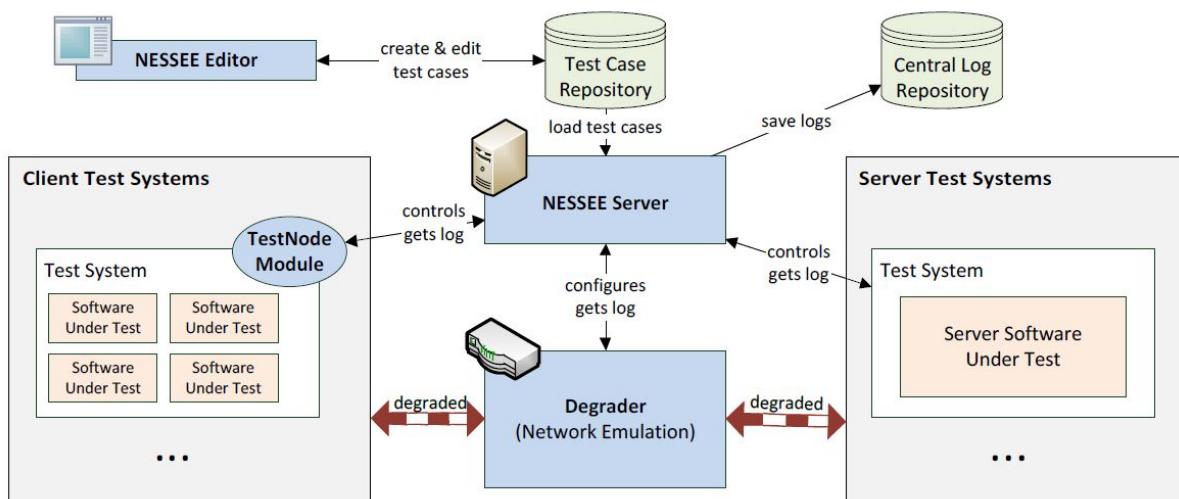


Abbildung 16: Architektur von NESSEE [Lübke et al., 2013]

## 2.3. NESSEE

---

Einen Überblick der Architektur von NESSEE liefert Abbildung 16. Die Testsysteme unterteilen sich zum Nachempfinden von verteilten Clients und Servern in *Client Test Systems* und *Server Test Systems*. Diese sind üblicherweise virtuelle Maschinen (VM), können aber auch physische Rechner darstellen. Auf den Systemen wird die zu testenden *Software Under Test (SUT)* installiert. Zwischen beiden Komponenten befindet sich der Degrader zur Emulation der Netzwerkparameter. Weiterhin koordiniert der zentrale NESSEE Server die Test Systeme, deren SUT und den Degrader. [Lübke et al., 2013] Ein Test wird durch einen *Test Case* beschrieben und durch die *Test Description Language (TDL)* definiert (Abbildung 17). Mit Hilfe eines Web-Interfaces lassen sich Tests erstellen, koordinieren und editieren. Weiterhin können Tests zur Laufzeit kontrolliert und manipuliert werden. Die Ergebnisse von abgeschlossenen Tests befinden sich zur Analyse im *Central Log Repository*. Das *Test Case Repository* verwaltet die verfügbaren Tests. [Lübke et al., 2013]

```
<NetworkCapabilities id="DSL">
    <delay direction="both" variation="7"
        distribution="laplace">12</delay>
    <loss direction="up">0.1</loss>
    <loss direction="down">0.09</loss>
    <reordering direction="both">0.3</reordering>
    <duplication direction="both">0.001</duplication>
    <disconnect start="24h" duration="5s" />
</NetworkCapabilities>
<NetworkCapabilities id="DSL6000" BasedOn="DSL">
    <bandwidth direction="down">6000</bandwidth>
    <bandwidth direction="up">1000</bandwidth>
</NetworkCapabilities>
<!-- ... -->
```

Abbildung 17: Einstellung von Netzwerkparametern in einem Test Case mittels TDL  
[Lübke et al., 2013]

In Testläufen mit hohen Nutzerzahlen kommt es zu einem hohen Datenaufkommen durch eine Vielzahl zu emulierende Software (SUT). Dadurch scheint das NESSEE Projekt wegen des hohen Datenaufkommens prädestiniert als Anwendungsfall für das in dieser Arbeit zu entwickelnde System. Es besteht die Möglichkeit während eines Tests verschiedene Daten wie Netzwerkparameter oder CPU-Last in einem ausgewählten zeitlichen Abstand zu sammeln. Später können die Daten durch das in dieser Arbeit entwickelte System ausgewertet werden. Methoden, wie das *Sampling* oder Filtern, sind bei umfassenden Testläufen unabdingbar, um dem großen Datenvolumen entgegenzuwirken. Weiterhin entlasten Visualisierungen den Verstehensprozess der Daten.

NESSEE entstand in Zusammenarbeit mit dem Lehrstuhl Rechnernetze und Citrix Online. Zur Zeit wird NESSEE von Citrix Online zum Testen von Videokonferenzsystemen und E-Collaboration verwendet.



## 3. Verwandte Arbeiten

Die verwandten Arbeiten teilen sich in vier relevante Abschnitte auf. Der erste Abschnitt behandelt Methoden und Systeme zur effizienten Verarbeitung von verteilten Daten (Abschnitt 3.1). Der zweite Abschnitt zeigt weitere Visualisierungsformen für zeitbasierte und multivariate Daten auf (Abschnitt 3.2). Abschnitt 3.3 differenziert verschiedene Visualisierungsformen für zeitbasierte Daten in Bezug auf Veränderungen des Kontextes bei neu hinzukommenden Daten. Weiterhin werden verwandte Systeme zur Zeitreihenanalyse (Abschnitt 3.4) aus unterschiedlichen Domänen betrachtet. Ein Fazit (Abschnitt 3.5) gibt einen zusammenfassenden Abschluss gewonnener Erkenntnisse.

### 3.1. Verteilte Verarbeitung von Daten

Operationen auf immensen Datenmengen erfordern neue Ansätze und Systeme. Trotz der steigenden Rechenleistung von Computern, müssen große Datenmengen für eine zeitlich akzeptable Verarbeitung notwendigerweise auf mehrere Rechner verteilt werden. Diverse Methoden und Systeme haben in der jüngsten Vergangenheit dazu beigetragen.

*MapReduce* ist ein von GOOGLE eingeführtes Programmiermodell zur effizienteren Datenverarbeitung. Inspiriert wurde das Modell durch die *map* und *reduce* Primitiven funktionaler Programmiersprachen wie Lisp. Die Verarbeitung nimmt eine Reihe von Schlüssel-Wert-Paaren entgegen und erstellt wiederum eine neue Reihe von Schlüssel-Wert-Paaren. Der Ablauf lässt sich in die zwei Schritte *Map* und *Reduce* aufteilen. *Map* nimmt die Reihe von Paaren entgegen und bildet nach Anordnungen des Nutzers eine neue Reihe aus Zwischenschlüsseln und Werten. Alle Werte mit dem selben Zwischenschlüssel werden in einer neuen Reihe zusammengefasst und als Zwischenschlüssel-Reihe-Paar der *Reduce* Funktion übergeben. *Reduce* wird abschließend zur Datenreduzierung benutzt, indem beispielsweise die Werte aus der Reihe des Zwischenschlüssel-Reihe-Paars zusammengefasst werden.

#### Beispiel 3.1 MapReduce

---

In einer Sammlung von Texten sollen Wörter gezählt werden. Die *Map* Funktion erzeugt das Wort als Zwischenschlüssel und eine Reihe von Zahlen für das Aufkommen des Wortes. In diesem Fall stehen in der Reihe nur Einsen (beispielsweise das Wort Visualisierung als [“Visualisierung”][1, 1, 1, 1, 1] für ein 5-faches Aufkommen in allen Texten). Die *Reduce* Funktion nimmt dieses Datum entgegen und summiert die einzelnen Aufkommen zu [“Visualisierung”][5].

### 3.1. Verteilte Verarbeitung von Daten

Das Programmiermodell kann für einzelne Rechner sowie große verteilte Systeme angewandt werden. Letzteres entfaltet erst das ganze Potenzial von *MapReduce*. Es ermöglicht die Verteilung und Parallelisierung benötigter Rechenkapazität. Einzelne Rechner können auch Prozessorkerne der Maschine einbinden und Prozesse parallelisieren. Das *MapReduce* Framework von Google kapselt die Details zur Parallelisierung, Fehlerbehandlung, Daten- und Lastenverteilung. [Dean and Ghemawat, 2008]

DRYAD ist die Antwort von MICROSOFT auf GOOGLES *MapReduce*. Es ist ein generelles Framework mit mehr als zwei Phasen und mehr Operationen als *Map* und *Reduce*. Weiterhin bietet es die Möglichkeit weitaus komplexere Verarbeitungsprozesse über einen azyklischen Graphen zu beschreiben. Kanten repräsentieren Datenkanäle wie TCP Verbindungen oder *Shared Memory* und Ecken stellen Operationen (Programme) dar. Der Verarbeitungsgraph wird später auf die physischen Daten abgebildet. Gegenüber *MapReduce* ist DRYAD ein vollständiges und komplexeres Framework. [Isard et al., 2007]

HADOOP [Shvachko et al., 2010] ist ein Framework für skalierbare und verteilte Systeme für das *batch processing* und der Analyse von großen Datenmengen. Es nutzt das *MapReduce* Programmiermodell zur Transformation der Daten, ermöglicht die zuverlässige Speicherung von großen Datenmengen und verschickt die Daten mit einer hohen Geschwindigkeit an Nutzeranwendungen. Die Skalierung durch mehrere tausend Rechnerknoten betrifft Rechenleistung, Speicherplatz und Ein- sowie Ausgabebandbreite. HADOOP ist ein Apache Projekt und besteht aus mehreren Komponenten unterschiedlicher Unternehmen. Die Kernkomponenten sind das *MapReduce* und HDFS (Hadoop Distributed File System). Letzteres speichert Metadaten und Anwendungsdaten auf zwei verschiedenen Knotentypen: *NameNode* und *DataNode*. Für eine erhöhte Zuverlässigkeit und mehrere Möglichkeiten der ortsabhängigen Rechenleistung werden die Anwendungsdaten auf einer Vielzahl von *DataNodes* repliziert. Ein *HDFS Client* wird für den Zugriff auf HADOOP von Nutzeranwendungen genutzt. Auf den Dateien und Ordnern werden übliche Create, Read, Update, Delete (**CRUD**) Operationen zugelassen. Neben *HDFS* besteht HADOOP noch aus weiteren Komponenten, welche unter anderem auch *MapReduce* benutzen:

- *Pig* ist für die Erstellung von *MapReduce* Abfragen zur Behandlung von unstrukturierten Daten.
- *Hive* ist eine Infrastruktur für *Data-Warehouses* zur langfristigen Analyse von Daten.
- *HBase* ist eine Spalten orientierte Datenbank.
- *ZooKeeper* ist ein verteilter Koordinationsservice.

SPARK wurde ähnlich zu HADOOP für die verteilte Verarbeitung und Analyse von großen Datenmengen entwickelt. Trotzdem existiert ein anderer Anwendungsfokus als bei HADOOP. Während HADOOP mit *MapReduce* für azyklische Analyseaufgaben prädestiniert ist, liegen die Stärken von SPARK in iterativen Aufgaben, sowie der interaktiven Analyse auf wiederverwendbaren Datensätzen. Viele Algorithmen wenden eine Funktion auf einen gleichen Datensatz an, um beispielsweise Parameter zu optimieren. In *MapReduce* Prozessen, wie bei HADOOP, würde jedes Mal der Datensatz neu aus dem Hauptspeicher geladen werden. SPARK nutzt *resilient distributed datasets (RDD)* zum *Caching* und Verteilen von *read-only* Daten. Beim ersten iterativen Laden werden die Daten wie in HADOOP aus dem Hauptspeicher geladen. Jede weitere iterative Aufgabe auf dem Datensatz erfolgt auf dem vielfach schnelleren *Cache*. Auch in der interaktiven Analyse bietet es sich an, einen zu analysierenden Datensatz in den *Cache* zu laden. In HADOOP würde jedes Mal ein separater *MapReduce* Prozess ausgeführt und die Daten aus dem Hauptspeicher geladen werden. Eine wissenschaftliche Arbeit zu SPARK zeigt auf, dass SPARK gegenüber HADOOP für iterative Analyseaufgaben 10 mal schneller ist und 39 GB Datensätze in weniger als einer Sekunde analysiert werden können. [Zaharia et al., 2010]

Das verteilte *Queueing System* KAFKA sammelt und verschickt große Volumen von Nachrichten als Datenströme. Durch die Skalierbarkeit ermöglicht KAFKA einen hohen Daten durchsatz. Weiterhin bietet das System eine API für die Bereitstellung von Nachrichten in Echtzeit. Anstelle eines Push-basierten Modells, wie es traditionelle Nachrichtensysteme verwenden, verschickt KAFKA die Nachrichten basierend auf einer Pull-Abfrage. Demnach wird das Problem der zu hohen Datenflut auf rechenschwachen Clients umgangen. Für Nachrichtenquellen besteht die Möglichkeit eine Nachrichtensammlung anstelle einzelner Nachrichten zu schicken. So werden bei einem hohen Datendurchsatz viele einzelne zeitaufwendige TCP/IP Übertragungen umgangen. Nachrichten werden sogenannten *Topics* zugeordnet. Nachrichtenquellen können demnach einem *Topic* neue Nachrichten zukommen lassen. Die Daten werden zur Lastenverteilung auf mehreren Servern (*Broker*) in Partitionen gespeichert. Ein Client kann nach dem *Publish-Subscribe-Modell* ein *Topic* abonnieren (Abbildung 18a). Für eine höhere Effizienz werden erst nach einer bestimmten Zeit oder nach einer konfigurierbaren Anzahl an Nachrichten die Partitionen bei den *Brokern* gespeichert. Auch erst dann werden die neuen Nachrichten der abonnierten *Topics* an Clients verbreitet. Zur Adressierung der einzelnen Nachrichten werden Offsets in der Länge einer Partition anstelle von Identifikationsnummern verwendet. Neben dem Hauptdatencenter existiert ein Analysedatencenter. Die dort ansässigen *Broker* holen Daten aus dem Hauptdatencenter ab, die dann zu den geografisch naheliegenden HADOOP und einem eigenen *Data-Warehouse* (DWH) zur Analyse und Auswertungen

### 3.1. Verteilte Verarbeitung von Daten

---

weitergeleitet werden (Abbildung 18b). KAFKA wird unter einer Last von hunderten Gigabytes neuer Daten pro Tag bei LINKEDIN für die *Activity Streams* und internen Datenverarbeitungen genutzt. [Jay Kreps, 2011]

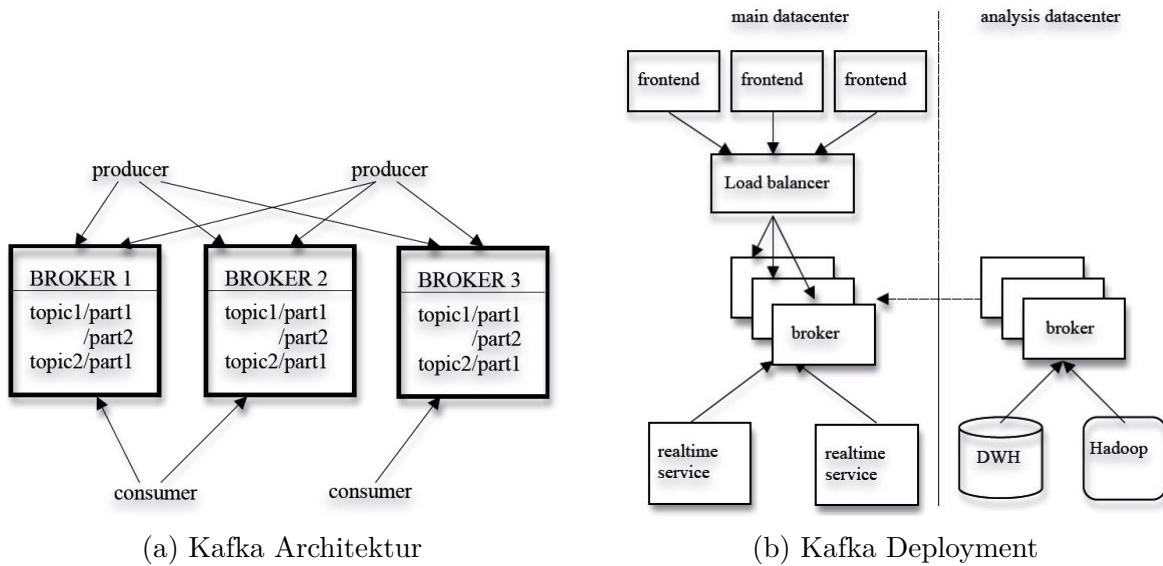


Abbildung 18: Kafka als Nachrichtensystem [Jay Kreps, 2011]

STORM ist ein verteiltes Echtzeitverarbeitungssystem. Gegenüber *batch processing* Systemen wie HADOOP, nutzt STORM das Prinzip des *realtime processing*. *Batch processing* ist zur Verarbeitung von großen Datenmengen, die über eine längere Zeitperiode gesammelt wurden, gedacht. *Realtime processing* hingegen ist ein stetig eintreffender Datenfluss, welcher verarbeitet und weitergeleitet wird. Demnach besteht ein möglicher Anwendungsfall für STORM in der Echtzeitanalyse von Daten. Das Konzept von STORM stellt sich wie folgt zusammen: Eine Topologie ist ein Graph bestehend aus *Spouts* und *Bolts* (Abbildung 19). *Spouts* ermöglichen die Integration von Datenquellen. Dies sind beispielsweise *Queueing Systems* wie KAFKA, aber auch Service-Schnittstellen wie die von TWITTER für die Datenübertragung. *Spouts* emittieren die Daten zu verteilten *Bolts* als *Streams*. *Streams* nutzen ein Schema für die Zusammenstellung der Wertetupel des Datenstroms und stellen den Datenstrom selbst dar. *Bolts* sind die Datenverarbeitungskomponente in STORM. Durchlaufende Daten werden hier gefiltert, aggregiert und zusammengeführt. Ein *Bolt* ist für eine simple Transformation zuständig, während mehrere *Bolts* komplexe Datenverarbeitungen durchführen können. *Bolts* und *Spouts* sind in der Lage mehr als einen *Stream* zu emittieren. [Storm]

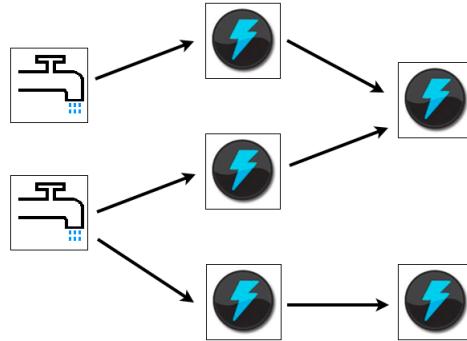


Abbildung 19: Storm Topologie aus Spouts (Wasserhahn) und Bolts (Blitze) [Storm]

DRILL [Hausenblas and Nadeau, 2013] und DREMEL [Melnik et al., 2010] sind interaktive, skalierbare ad-hoc *query systems*. Letzteres ist eine Variante von GOOGLE auf dessen Prinzip später das Open Source Projekt DRILL entstand. DRILL ist ein verteiltes System für die ad-hoc Analyse von Daten mit geringen Verzögerungszeiten. Es bietet Nutzern die Unterstützung zur Formulierung von Abfragen, um interaktiv mit einem großen Datensatz zu interagieren. Die Herausforderung für das *Extrahieren, Transformieren und Laden* (ETL) von Daten besteht in den verschiedenen Datenquellen und der performanten Datenverarbeitung sowie Analyse. Zwei Hauptprinzipien von DREMEL dienen als Grundlage für DRILL:

- Eine verschachtelte, spalten-orientierte Darstellung der Datenbank zur Speicherung von Daten.
- Ein skalierbarer, verteilter Algorithmus zur parallelen Verarbeitung einer Abfrage auf mehreren Maschinen.

DRILL verwendet dafür eine Architektur bestehend aus Nutzerschicht, Verarbeitungsschicht und Datenschicht (Abbildung 20). Die Nutzerschicht bietet dem Nutzer einige Schnittstellen (Kommandozeile, REST Interface, etc.) zur Ausführung der Abfragen. Die Verarbeitungsschicht beinhaltet mehrere erweiterbare Abfragesprachen, den Abfrageplaner, Ausführungs- und Speicherkomponenten. Abschließend können mehrere Datenquellen (HADOOP, MONGODB, CASSANDRA etc.) ebenfalls auf Basis von Plugins integriert werden. DRILL kann als Abfrageschicht für interaktive ad-hoc Analysen in größeren Systemkompositionen hinzugefügt werden.

### 3.1. Verteilte Verarbeitung von Daten

---

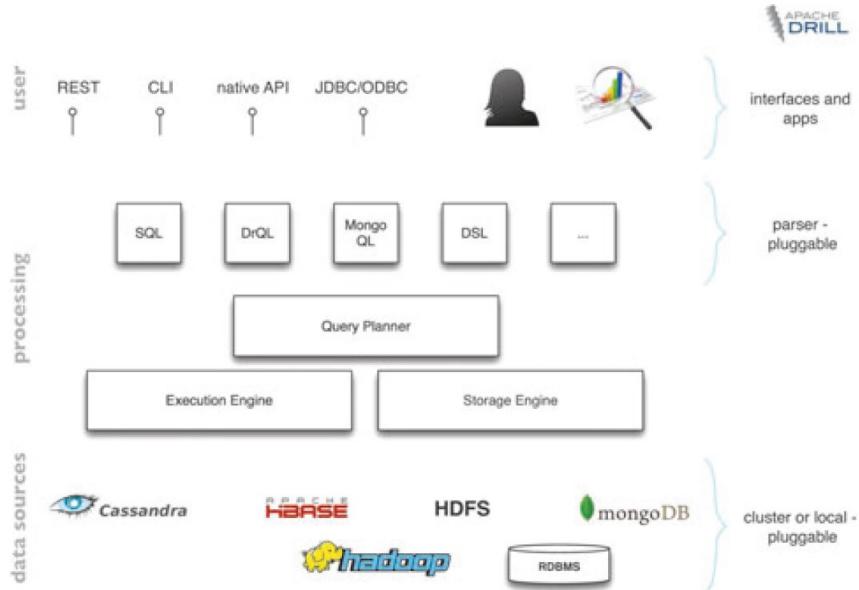


Abbildung 20: Apache Drill Architektur [Hausenblas and Nadeau, 2013]

IMPALA ist ähnlich zu DRILL und DREMEL ein weiteres Open Source Projekt für verteilte, interaktive Abfragen initiiert von CLOUDERA. Das System wird nativ in HADOOP genutzt, soll aber keine Ablösung zu Komponenten wie HIVE oder dem generellen Programmiermodell *MapReduce* darstellen. Genau wie DRILL und DREMEL ist IMPALA für ad-hoc Abfragen von Nutzern gedacht, welche in kürzester Zeit bearbeitet werden müssen. Komponenten wie HIVE und *MapReduce* zielen dagegen auf umfassendere Analyseverfahren ab. [Kornacker and Erickson, 2012]

Unter all den vorgestellten Datenverarbeitungssystemen, soll auch ein kurzer Überblick über aktuelle NoSQL Datenbanken geliefert werden. NoSQL Datenbanken basieren auf dem Konzept der nicht-relationalen Speicherung von Daten. Gegenüber relationalen Daten, welche die Daten in einer zweidimensionalen Tabelle abspeichern, verwenden NoSQL Datenbanken variierende Schemata von *document-oriented* (MONGODB) über *key/value* (REDIS) bis hin zu *Big Table* (HBASE aus HADOOP). NoSQL Datenbanken versprechen gegenüber relationalen Datenbanken einen schnelleren Zugriff auf die Daten und eine höhere Skalierung im Gegenzug für eine geringere Transaktionssicherheit. Daher sind sie vor allem für große Datenmengen geeignet. [Edlich et al., 2010]

MONGODB ist eine skalierbare *document-oriented* Datenbank. Ein Dokument lässt sich

praktisch auf ein Objekt aus der objektorientierten Programmierung abbilden. Demnach kann das Dokument, im Gegensatz zu einer *Row* von relationalen Datenbanken, auch hierarchische Strukturen annehmen. Das *Sharding* Konzept ermöglicht es, Daten über mehrere Server zu verteilen. Auch verwendet MONGODB das *MapReduce* Programmiermodell für flexible Zusammenführung und Reduzierung der abgefragten Daten. [Chodorow and Dirolf, 2010]

REDIS<sup>1</sup> ist eine verbreitete Schlüssel-Wert-Datenbank der NoSQL Datenbankfamilie. Die Datenbank nutzt für eine erhöhte Performanz den Arbeitsspeicher zur Speicherung der Datensätze. Die Eigenschaften der Datenstruktur und des Speichermediums heben REDIS deutlich von anderen NoSQL Datenbanken ab. Eine hohe Performanz wird hierbei gegen eine einfache Datenstruktur und absehbaren Speichervolumen eingetauscht.

APACHE CASSANDRA<sup>2</sup> wird für eine performante Verwaltung von großen Datenmengen über verteilte Server verwendet. Es gehört zu den *column-oriented* Datenbanken und bietet hohe Skalierbarkeit und Ausfallsicherheit. Ein Vorteil der Speicherung von großen Datenmengen in Spalten liegt im Zusammenhang mit der Datenanalyse. Einfache Operationen auf eine Spalte wären hier beispielsweise die Berechnung von Summen, Durchschnittswerten und Minimal- beziehungsweise Maximalwerte. In relationalen Datenbanken würde dies eine Reihe von sequentiellen Lesezugriffen auf einzeln indizierte Objekte bedeuten. Erst kürzlich wurde bekannt, dass die Firma hinter CASSANDRA eine Partnerschaft mit DATABRICKS, dem Betreiber von SPARK einging.<sup>3</sup> Die Zusammenarbeit beinhaltet vor allem eine engere Verknüpfung zwischen CASSANDRA als verteilten Datenspeicher und SPARK als Verarbeitungssystem für effiziente Dateneinblicke.

Auch verschiedene Programmiersprachen sind für die effiziente Verarbeitung von Daten zu betrachten. Statistische Programmiersprachen wie R oder das jüngste JULIA werden zur fortgeschrittenen Analyse von Daten genutzt. Die Entwickler von R wurden von den Programmiersprachen S und SCHEME inspiriert. Hierbei wurde eine Möglichkeit gesucht, die Stärken beider Sprachen zu kombinieren. Während die Herleitungen der Implementierung und des semantischen Aufbaus aus SCHEME stammen, hat R dessen ungeachtet die äußerliche Erscheinung von S. [Ihaka and Gentleman, 1996] R wird heutzutage von über zwei Millionen Analytikern aus unterschiedlichsten Domänen weltweit verwendet.<sup>4</sup> Mittlerweile bieten Erweiterungspakete auch die Möglichkeit zur fortgeschrittenen Datenvisualisierung in R an.<sup>5</sup>

---

<sup>1</sup><http://redis.io/>, Stand: 23.07.2014

<sup>2</sup><http://cassandra.apache.org/>, Stand: 23.07.2014

<sup>3</sup><http://databricks.com/blog/2014/05/08/Databricks-and-Datastax.html>, Stand: 24.07.2014

<sup>4</sup><http://ReadWrite.com/2011/09/07/unlocking-big-data-with-r>, Stand: 24.07.2014

<sup>5</sup><http://www.r-project.org/>, Stand: 24.07.2014

### **3.2. Fortgeschrittene Visualisierungsformen für zeitbasierte, multivariate Daten**

---

JULIA ist eine dynamische Programmiersprache mit den Anforderungen eine hohe Performance für numerische Programme zu liefern. Die Sprache ist noch sehr jung (2012), zeichnet aber in Benchmark Tests (Abbildung 21) eine Überlegenheit gegenüber anderen Programmiersprachen ab. [Bezanson et al., 2012]

test	Julia	Python	MATLAB®	Octave	R	JavaScript
fib	1.97	31.47	1336.37	2383.80	225.23	1.55
parse_int	1.44	16.50	815.19	6454.50	337.52	2.17
quicksort	1.49	55.84	132.71	3127.50	713.77	4.11
mandel	5.55	31.15	65.44	824.68	156.68	5.67
pi_sum	0.74	18.03	1.08	328.33	164.69	0.75
rand_mat_stat	3.37	39.34	11.64	54.54	22.07	8.12
rand_mat_mul	1.00	1.18	0.70	1.65	8.64	41.79

Abbildung 21: Benchmark mathematischer Berechnungen im Vergleich zu C (C hat einen Wert von 1) [Bezanson et al., 2012]

Die in diesem Abschnitt dargelegten Technologien können in Analysesystemen für große Datenmengen oftmals ineinander verwoben werden. HADOOP ermöglicht die länger andauernde Analyse von massiven Datenaufkommen durch das *MapReduce* Programmiermodell. Alternativen, mit einem anderen Aufgabenfokus, wie SPARK, sind für iterative Echtzeitanalyseaufgaben zuständig. Weiterhin sind stetig genutzte Systeme wie HADOOP in Nachrichtensystemen wie KAFKA integrierbar. KAFKA bewerkstellt das verteilte *Queueing* von großen Mengen an Echtzeitdaten zur Lastenverteilung. Systeme wie DRILL und IMPALA geben ähnlich wie SPARK die Möglichkeit interaktive Analyseaufgaben in nahezu Echtzeit zu lösen. Abschließend werden oftmals NoSQL Datenbanken für die Speicherung der großen Datenmengen verwendet. Bei größeren Datenaufkommen, wie in der Definition von *Big Data*, wird häufig zu Lösungen wie HADOOP migriert.

## **3.2. Fortgeschrittene Visualisierungsformen für zeitbasierte, multivariate Daten**

Am einfachsten wäre es zeitbasierte Daten auf eine einheitliche Visualisierungsform wie die *Line Graphs* abzubilden. Im praktischen Visualisierungsumfeld ist dies nicht immer der Fall. Vor allem durch das Aufkommen von multivariaten Daten müssen weitere Visualisierungsformen hervorgebracht werden. Aus verschiedenen Domänen stehen eine Vielzahl von Daten in Beziehung zur Zeit: Börsenkurse oder Vitalwerte von Patienten in einem Krankenhaus, aber auch Fotoalben, Projektplanungen und Zeitschriftenartikel.

Dafür existieren eine Vielzahl von Darstellungen für die unterschiedlichen Anforderungen. Börsenkurse können mit *Flocking Boids* [Moere, 2004] (Abbildung 22a), Zeitschriftenartikel mit einem *ThemeRiver* [Havre et al., 2002] (Abbildung 22b) und Projektplanungen mit *PlanningLines* [Aigner et al., 2005] (Abbildung 22c) visualisiert werden. [Aigner et al., 2007]

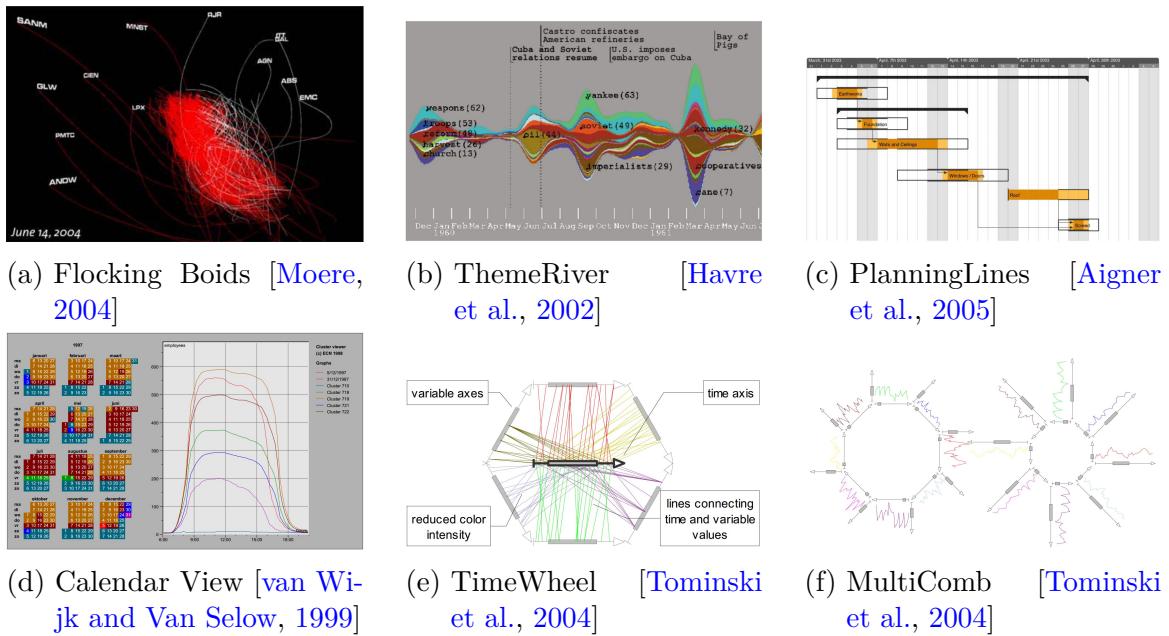


Abbildung 22: Visualisierungsformen für zeitbasierte Daten

Die *Flocking* Methode gilt als Grundlage für die *Flocking Boids*. Sie wurde von Proctor und Winter hervorgebracht [Proctor and Winter, 1998] und beruht auf den visuellen Variablen Farbe und Bewegung. In der Arbeit von Proctor und Winter wurden Fische im 3D-Raum über einen Zeitraum visualisiert. Die Farbe stellt dabei ein weiteres Attribut der Daten dar. Moere [Moere, 2004] griff diese Methode auf, um Börsenkurse als *Flocking Boids* zu visualisieren. Die Bewegung resultierte dabei aus den Kursschwankungen.

Der *ThemeRiver* [Havre et al., 2002] beruht auf den Grundlagen der *Stacked Graphs*. Hier markieren einzelne eingefärbte Strömungen Themen. Ihre Breite stellt die Relevanz des Themas dar. Zeitlich betrachtet können so praktisch relevante Themen zu bestimmten Zeitpunkten oder Zeitintervallen ausgemacht werden.

*PlanningLines* [Aigner et al., 2005] machen es sich zur Aufgabe, der zeitlichen Ungewissheit in Domänen wie Projektplanung und medizinischen Versorgung entgegenzutreten. Die *PlanningLine* beschreibt eine Aktivität mit den Eigenschaften: *earliest starting time*

### **3.3. Echtzeitvisualisierung und Änderungen des Kontextes**

---

(EST), *latest starting time* (LST), *earliest finishing time* (EFT) und *latest finishing time* (LFT) sowie die daraus resultierenden *maximum duration* und *minimum duration* der Aktivität. Die einzelnen Aktivitäten werden unterhalb einer Zeitachse angeordnet.

In der Literatur existieren eine Vielzahl weiterer Visualisierungsformen wie die *Calendar View* [van Wijk and Van Selow, 1999] (Abbildung 22d), das *TimeWheel* [Tominski et al., 2004] (Abbildung 22e) und die *MultiComb* [Tominski et al., 2004] (Abbildung 22f) für die Darstellung (multivariater) zeitbasierter Daten.

## **3.3. Echtzeitvisualisierung und Änderungen des Kontextes**

Die Ausarbeitung von Keim et al. [Krstajic and Keim, 2013] beschäftigt sich mit verschiedenen Formen der Echtzeitvisualisierung und ihrem Verhalten beim Hinzunehmen neuer Daten. Dabei wird davon ausgegangen, dass die bereits vorhandenen visualisierten Daten in der Anzahl größer sind als die neu hinzukommenden Daten. In einigen Teilen der Ausarbeitung wird auch von einem einzelnen Datenpunkt ausgegangen der hinzugefügt wird. Für ihre Ausarbeitung stellten sie sich folgende Fragen:

*“How should the visualization change when we add new data? Does the whole layout have to be recomputed when we add just one element like in force-directed graphs, or we can easily add it like in scatterplots? But what if the new attribute value is out of current minimum/maximum ranges? Can i identify what is new and where did my old data go?”*

Dabei sei zu unterscheiden, ob die Änderungen innerhalb oder außerhalb des Wertebereich vom vorhandenen Datensatz erfolgt. Die Ausarbeitung beschäftigt sich mit *Streamgraphs* (*Stacked Graphs*), *Line Graphs*, *Horizon Graphs* und Pixel-basierten Visualisierungen (Abbildung 23). Daneben werden auch *Scatterplots*, *Treemaps* und *Word Clouds* erörtert.

Achsenbasierte Visualisierungen sind gleichermaßen von neu hinzukommenden Daten betroffen. Horizontal existieren zwei Möglichkeiten: Im ersten Fall können alte Daten in der Zeitspanne gestaucht werden, um den gesamten Datensatz in der Visualisierung zu behalten. Der zweite Fall würde den gesamten Datensatz um die neu hinzukommende Zeit horizontal verschieben. Daten, die nicht mehr in der Zeitspanne liegen, würden entfernt werden. Vertikal besteht die Möglichkeit, dass neue Daten außerhalb des festgelegten Wertebereichs liegen. Demnach müsste die vertikale Skalierung neu angepasst werden. Keim et al. sehen diese Verschiebungen und Skalierungen als vernachlässigbar. *Streams* in *Streamgraphs* folgen zum besseren Verständnis der Visualisierung einer bestimmten Anordnung. Während *Streams* mit wenigen Schwankungen mittig angeordnet

	What can change?	Loss of context
<b>streamgraph</b>	<ul style="list-style-type: none"> <li>- Ranges of the axes</li> <li>- Streams reordered</li> <li>- Streams added/removed from the screen</li> </ul>	<ul style="list-style-type: none"> <li>- significant due to stream reordering</li> </ul>
<b>line chart(s)</b>	<ul style="list-style-type: none"> <li>- Ranges of the axes</li> <li>- Lines added/removed from the screen</li> </ul>	<ul style="list-style-type: none"> <li>- only when outside min/max y-axis range</li> <li>- other significant problems: overplotting, clutter, low resolution</li> </ul>
<b>horizon chart(s)</b>	<ul style="list-style-type: none"> <li>- Ranges of the axes</li> <li>- Color ranges for new max/min values</li> <li>- Number of charts</li> <li>- Charts reordered</li> </ul>	<ul style="list-style-type: none"> <li>- change of color (when values are out of range)</li> <li>- chart reordering</li> </ul>
<b>pixel-oriented (recursive pattern)</b>	<ul style="list-style-type: none"> <li>- Pixels added/removed from the screen</li> <li>- Color change</li> <li>- Dimension reordering</li> <li>- Recursion levels changed</li> </ul>	<ul style="list-style-type: none"> <li>- color change when values outside range</li> <li>- if dimension reordering needed (rare)</li> </ul>

Abbildung 23: Ausschnitt eines Überblicks von Änderungen durch Echtzeitdaten in verschiedener Visualisierungsformen und der einhergehende Verlust des Kontextes [Krstajic and Keim, 2013]

werden, finden *Streams* mit häufigen Schwankungen außen Platz. Bei Echtzeitdaten lässt sich vorher nicht festlegen, welche *Streams* größeren Schwankungen ausgesetzt sein werden. Nach dem Hinzufügen eines neuen Datums können die *Streams* nach den Bestimmungen neu angeordnet werden. Auf der einen Seite könnte dies für Verwirrung beim Nutzer sorgen, auf der anderen Seite müsste die gesamte Visualisierung rechenintensiv neu gezeichnet werden. Die zweite Möglichkeit wäre es, die *Streams* in ihrer initialen Anordnung zu belassen. Dies würde die optimale Darstellung dieser Visualisierungsform gefährden. Das Fazit von Keim et al. bezüglich der Echtzeitvisualisierung in *Streamgraphs*: “*the streamgraph can hardly be a good choice for streaming data.*” [Krstajic and Keim, 2013]

*Horizon Graphs* und *Line Graphs* unterliegen genau wie *Streamgraphs* der Achsenbasierten Veränderungen. *Horizon Graphs* müssen hinzukommend die Einteilung der Bänder neu berechnen. Für *Line Graphs* wäre der Verlust des Kontextes vernachlässigbar klein, weswegen diese Art der Visualisierung zu für Echtzeitdaten zu bevorzugen sei. In Pixel-basierenden Visualisierungen würden neue Daten, die außerhalb des Wertebereichs liegen, eine Neueinfärbung der gesamten Pixel mit sich ziehen. Dies würde eine rechenintensive Neuberechnung und Darstellung der Visualisierung beinhalten.

### 3.4. Systeme zur Datenexploration von zeitbasierten Daten

In diesem Abschnitt werden verwandte Systeme zur Echtzeitvisualisierung von großen Datenmengen vorgestellt. Hierbei geht es vor allem um die einzelnen Komponenten zur Datenanalyse, Reduzierung der Daten, Visualisierung und Echtzeitfähigkeit. Nicht alle hier vorgestellten Systeme beinhalten alle dargelegten Komponenten.

Wegbereiter für die Analyse von zeitbasierten Daten stammen oftmals aus der medizinischen Domäne [Goren-Bar et al., 2004, Kostkova et al., 2014] oder allgemeiner aus dem Bereich des Monitoring [Fischer et al., 2012, Shabtai et al., 2006, Stoffel et al., 2013]. In diesen Gebieten ist es vor allem wichtig, Anomalien in Zeitreihen auszumachen. Während dies in der medizinischen Domäne Schwankungen von Vitalwerten der Patienten sein können, sind es bei der Überwachung von Systemen oft Abweichungen die aufgedeckt werden müssen.

KNAVE-II ist ein Interface zur Visualisierung von zeitbasierten Daten. Es unterstützt die Bereitstellung von abstrahierten Informationen auf Basis einer domänen spezifischen Wissensbasis und zeitlichen Mustern. Weiterhin können die abstrahierten und rohen Daten durch Interaktionen erkundet werden. Ein verteiltes System stellt die verschiedenen Komponenten zur Abstraktion der Daten auf Basis von zeitlichen Mustern und domänen spezifischen Wissen bereit. KNAVE-II wurde für den medizinischen Bereich, vor allem zur Überwachung von chronischen Patienten, entwickelt. Mittlerweile ist das System eine Grundlage für viele andere Anwendungen aus unterschiedlichsten Domänen. [Goren-Bar et al., 2004]

VISITORS (Visualization and Exploration of Multiple Time-Oriented Records) basiert auf KNAVE-II und widmet sich der Analyse zeitbasierter Daten für die Sicherheit von Netzwerken. Es unterstützt Administratoren dabei, abweichendes Verhalten in Netzwerk und Infrastruktur aufzudecken. Zeit, die durch automatische Analysen gespart werden kann, ist bei Angriffen auf Systeme und Infrastruktur essentiell. Demnach reicht die alleinige Darstellung von rohen Daten oftmals nicht aus. VISITORS nutzt eine “*knowlegde-based temporal abstraction (KBTA)*” Methode für eine automatische Herleitung von kontextspezifischen Interpretationen. Die *knowledge-base* für VISITORS umfasst Wissenselemente aus der IT-Sicherheit, die von Experten aus dem Bereich bereitgestellt werden. Ein Wissenselement wäre beispielsweise “*a period of 5 hours, during the night, of a high number of FTP connections within the context of No User Activity, which might indicate the existence of a Trojan in the computer*”. [Shabtai et al., 2006] VISITORS unterstützt die Visualisierung und Interaktion von einzelnen und gruppierten Datenobjekten auf mehreren (rohe Daten, interpretierte Daten) Abstraktionsebenen. [Shabtai et al., 2006]

MEDI+BOARD ist der medizinischen Domäne zuzuordnen, genauer dem Bereich der *Epi-*

*demic Intelligence.* Es dient der Prävention und Kontrolle von ansteckenden Krankheiten. Während es normalerweise mehrere Wochen dauern würde, Datensätze für aufkommende Krankheiten zu analysieren, soll MEDI+BOARD die Analyse der Daten in kurzer Zeit bewerkstelligen. Online Nachrichten, klinische Befunde, medizinische Plattformen und weitere Quellen werden hierbei als Datenquelle genutzt. Die Echtzeitdatenkanäle können auf Krankheitssignale untersucht, mit Signalen aus anderen Kanälen korreliert (Kreuzvalidierung) und in einem *Public Health Dashboard* dargestellt werden. Die Anwendung präsentiert sich als vollständig generisch. Demnach können austauschbare *Event Templates* für bestimmte Krankheitssignale unter Zuhilfenahme von medizinischen Experten erstellt werden. Positive Signale, die sich auf einem *Event Template* abbilden lassen, werden neben weiteren Rohdaten zum *Public Health Dashboard* weitergeleitet. Als Darstellung dienen eine Vielzahl von interaktiven Echtzeitvisualisierungen wie *Bar Graphs*, *Maps* und *Tables* (Abbildung 24b). *Line Graphs* stellen die zeitbasierten Daten der verschiedenen Kanäle dar. Die Anwendung ist als Dienst unter WINDOWS AZURE implementiert (Abbildung 24a). Dies ermöglicht die dynamische Skalierung relativ zu den eintreffenden Daten. Weiterhin wird das *MapReduce* Prinzip verwendet, um die benötigte Rechenleistung auf mehrere Knoten zu verteilen. Das *Public Health Dashboard* ist eine auf HTML5 und JavaScript basierende Implementierung. [Kostkova et al., 2014]



Abbildung 24: medi+board [Kostkova et al., 2014]

TIMESEARCHER ist eine Analyseanwendung für mehrere Zeitserien. Als Hauptvisualisierung werden Zeitserien in einem *Multi-Series Line Graph* dargestellt. Weitere Visualisierungen wie *Tables* und *Small Multiples* von Zeitreihen sind mittels *Linking* und *Brushing* untereinander verknüpft. Weiterhin existieren *Tooltips* für einzelne Datenobjekte. Das

### *3.4. Systeme zur Datenexploration von zeitbasierten Daten*

---

Hauptaugenmerk der Anwendung ist das Ausfindig machen von Mustern. Dafür entwickelten Hochheiser et al. die *Timeboxes*. *Timeboxes* sind Rechtecke, die auf Zeitgraphen gezeichnet, verschoben oder in der Größe angepasst werden können. Die Rechtecke agieren als Filter auf dem festgelegten Bereich der Zeitreihen und grenzen diesen in eine Zeitspanne und Wertespanne ein. Datenobjekte, welche nicht in den Zeitbereich (x) und den Wertebereich (y) fallen, werden ausgefiltert. Mehrere *Timeboxes* können individuell oder gleichzeitig transformiert werden. Die simultane Transformation erlaubt es Teilmuster eines komplexen Musters durch Verschieben oder Skalieren der Rechtecke zu entdecken. Ein Nachteil birgt der unübersichtliche *Multi-Series Line Graph* bei einer großen Anzahl von Zeitserien. Weiterhin wird die Performanz bei *Timeboxes* über eine große Menge von Datenpunkte stark beansprucht. Als Lösung wurde eine zweite Ansicht mit einer niedrigeren Auflösung eingeführt. [Hochheiser and Shneiderman, 2004]

Das *Visual Analytics* System von Keim et al. [Stoffel et al., 2013] ist für die Analyse von Zeitserien entwickelt worden. Durch analytische Modelle können Anomalien oder Korrelationen zwischen Serien aufgedeckt werden. Vor allem in der Überwachung von Systemen und Infrastruktur ist dies ein wichtiger Schritt zur frühzeitigen Erkennung von Irregularitäten. Das System teilt sich in zwei Hauptbereiche: einem *stand-alone Server* und einem *rich client*. Der Server ist für die Verwaltung und Persistenz der Zeitserien, sowie für Funktionalitäten zur Analyse, Datenbeschaffung von verschiedenen Quellen und Bereitstellung der Daten zuständig. Daneben ist der Client für die Visualisierung und Interaktion mit den Zeitreihen verantwortlich (Abbildung 25a). Beide Komponenten kommunizieren über eine Remote Method Invocation ([RMI](#)) Schnittstelle. Das Interface ist weitestgehend generisch. Weiterhin benutzt die Datenbank *Sampling* und Füllmethoden, um fehlende Werte in einer Zeitreihe zu erzeugen und die Konsistenz zwischen Zeitreihen zu wahren. Später vereinfacht dies den Vergleich der (fouriertransformierten) Zeitserien. Der Client ist eine *NetBeans Rich Client Platform*. Das graphische Interface besteht aus mehreren Teilbereichen zur Manipulation, Darstellung und kontextsensitiven Anzeigen sowie Kontrollementen. Für die Zeitserien werden *Small Multiples* in einer vertikalen Ausrichtung genutzt (Abbildung 25b). Die Begründung: “*In the Western world, people are used to read text and charts from the left to the right. This is also the case for line charts. This leads to the behavior, that viewers tend to follow a single line chart, instead of comparing them to each other even if there are multiple charts drawn next to each other. By placing the time axis not on the horizontal, but on the vertical, we force the viewer to break this habit, and try to direct the perception to comparing different line charts.*” Die hohe Performanz des Systems wird durch ein Beispiel hervorgehoben. Demnach wird eine Abfrage auf einen Datensatz mit 1,1 Million Zeitserien (10 Monate umfassend mit 5 Minuten Abständen der Datenpunkte) in einer Minute bewerkstelligt (Intel Core 2 Quad Prozessor, 8 GB RAM, Intel X-18 SSD).

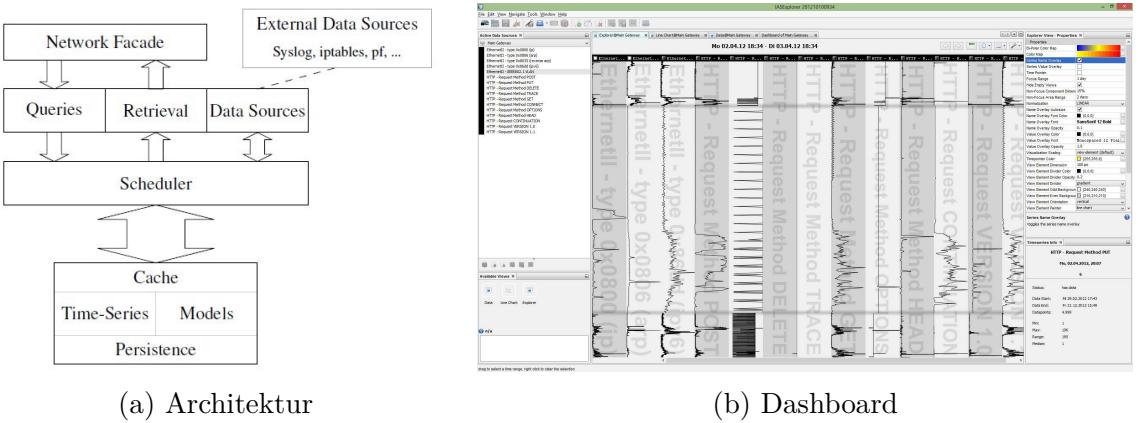


Abbildung 25: Visual Analytics Tool von Keim et al. [Stoffel et al., 2013]

Der LINE GRAPH EXPLORER adressiert eine ähnliche Motivation wie das *Visual Analytics* System von Keim et al. Auch hier wird versucht eine möglichst kompakte Visualisierungsform für eine große Menge an Zeitserien für das System zu finden. LINE GRAPH EXPLORER unterteilt die Visualisierung in einen Overview- und Detail-Bereich. Die Übersicht fasst alle Zeitserien in einer vertikalen Ausrichtung zusammen (Abbildung 26a). Hier wird anstelle von *Line Graphs* eine auf Pixel-basierende kompaktere Visualisierungsform gewählt. Zu jedem Zeitpunkt markiert die Sättigung der Farbe eines Pixels den zugehörigen Wert. Durch das Markieren einer Zeitreihe werden im Detail-Bereich die zugehörigen *Line Graphs* dargestellt (Abbildung 26b). [Kincaid and Lam, 2006]

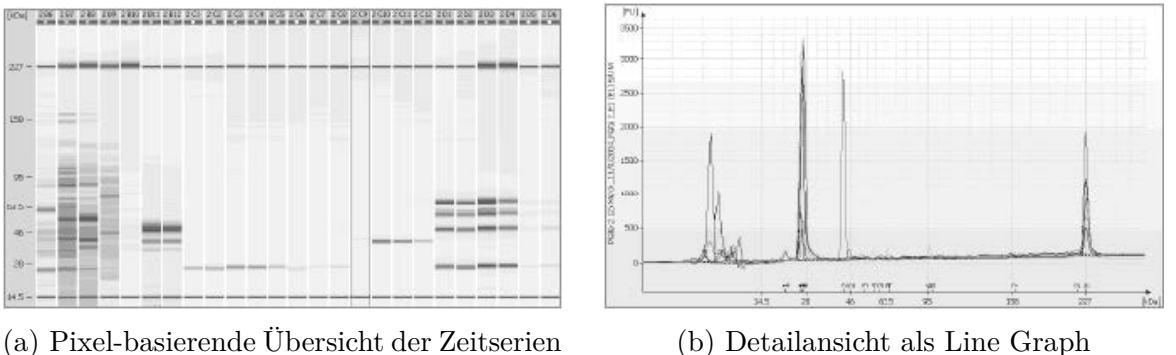


Abbildung 26: Line Graph Explorer [Kincaid and Lam, 2006]

Eine Vielzahl von Programmen und Systemen produzieren Statusnachrichten. Der EVENT VISUALIZER ermöglicht die Darstellung und Analyse dieser textuellen Echtzeitdaten. Da-

### *3.4. Systeme zur Datenexploration von zeitbasierten Daten*

---

bei handelt es sich um eine generische Verarbeitungs- und Analysearchitektur. Das System ist modular aus Komponenten zum Sammeln, Verarbeiten, Analysieren und Visualisieren dynamischer Datenströme zusammengestellt. Die Komponenten kommunizieren über einen *Java Message Service*. Der *Event Service* empfängt eingehende Daten und registriert neue Datenquellen. Die ankommenden Daten können durch einen oder mehrere *Analyzer* angereichert und für spätere Abfragen mit statistischen Werten versehen werden. Der *EVENT VISUALIZER* macht die bisher gesammelten und neu hinzukommende Daten sichtbar. Dafür werden auf Pixel-basierende Visualisierungen für die Darstellung der Events genutzt. Weiterhin ist die Grafik mit einem *Zooming* und *Panning* ausgestattet. [Fischer et al., 2012]

SPLUNK ist eine Anwendung zur Suche und Analyse von großen Mengen textueller Zeitserien mit Hilfe des *MapReduce* Programmiermodells. Es unterstützt Experten aus unterschiedlichen Domänen für Anwendungsverwaltungen, Sicherheitssysteme und Businessanalysen. Die *Splunk Search Language* ermöglicht das Ausführen der *Map* und *Reduce* Funktionen, ohne Kenntnisse über deren Aufteilung oder Programmierung zu haben. SPLUNK stellt dafür ein Vielzahl von zusammenstellbaren Kommandos bereit. Beispielsweise gibt die einfache Suchanfragen `search googlebot` alle Events mit der Textstelle *googlebot* zurück. Die Suchanfragen können durch eine Komposition aus Suchelementen auch weitaus komplexer ausfallen. Die wiedergegebenen Resultate können weiterhin grafisch aufbereitet in einem Dashboard platziert werden. Die Events werden in einem minimalen Datenschema gespeichert. Dies beinhaltet einen Text, Zeitstempel, Quelle sowie Quellentyp und dem Urheber. Daher sind die Daten schon Sekunden nach der Speicherung für die Suche auffindbar. Auch auf einem einzelnen Rechnerknoten nutzt SPLUNK die *MapReduce* Funktionen. Hier werden die Daten in Zeitspannen aufgeteilt auf denen die Funktionen unabhängig angewandt werden. [Sorkin, 2011]

TABLEAU SOFTWARE verfolgt das Konzept den Analyseprozess von Daten in Visualisierungen zu verpacken. Die Software umfasst entsprechend einem *Visual Analytic* Programm mehrere Visualisierungsformen in denen verschiedenste Daten eingefügt werden können. Die Datenschicht ermöglicht eine flexible Wahl der Datenarchitektur. Unter anderem ist die Integration von HADOOP und eigenen Datenbanken möglich. In TABLEAU existieren zwei Ansätze zur Dateninteraktion zwischen denen Nutzer wählen können: *Live connection* oder *In-memory*. Anstelle die Daten zu importieren, werden bei der *live connection* Abfragen an die externe Datenarchitektur gestellt. So können essentielle Verarbeitungsprozesse, wie die von HADOOP beispielsweise, verwendet werden. Die *in-memory* Lösung, welche für die Datenanalyse optimiert ist, bietet es an externe Datensätze zu importieren. TABLEAU ist eine kommerzielle *Visual Analytics* Lösung zur Datenanalyse und Datenvisualisierung mit der Möglichkeit einer Integration der eigenen Datenarchitektur. [Rueter and Fields, 2012]

### 3.5. Fazit

Alle aufgezeigten verwandten Arbeiten zeigen Relevanz zum gesuchten System zur Echtzeitvisualisierung von großen Datenmengen. Jedoch erfüllt keines der vorgestellten Projekte alle Kriterien der in dieser Arbeit geforderten Faktoren. Trotzdem konnten wichtige Erkenntnisse für die eigene Konzeption und Implementierung gewonnen werden.

Abschnitt 3.1 zeigt Ansätze zur effizienten Verarbeitung von verteilten Daten. *MapReduce* ist ein Programmiermodell, welches in vielen Programmiersprachen verfügbar ist und für die gesuchte Anwendung zur effizienten Verarbeitung bedacht wird. HADOOPS Architektur und *batch processing* für große Datenvolumen lieferte weitere relevante Einblicke in Systeme zur Verarbeitung großer Datenmengen. Daneben bietet SPARK weiteres Verständnis zur schnelleren Datenverarbeitung immenser Volumen durch *Caching*. Gegenüber dem *batch processing* für große Datenaufkommen von HADOOP, nutzt STORM ein *realtime processing* für die Verarbeitung eines stetig eintreffenden Datenflusses. Auch diese Art der Datenverarbeitung ist relevant für das gesuchte System zur Analyse von Datenströmen. DRILL, DREMEL und IMPALA bieten dagegen Einblicke in die effiziente ad-hoc Analyse von Daten mit geringen Verzögerungszeiten. Abschließend zeigen die NoSQL Datenbanken MONGODB, REDIS und CASSANDRA unterschiedliche Anwendungsfälle für große Datenmengen auf, welche auch im späteren Verlauf für die Implementierung evaluiert werden sollen.

Abschnitt 3.2 befasst sich mit weiteren, über den Grundlagen hinaus möglichen, zeitbasierten Visualisierungsformen, welche für die Konzeption und Implementierung von Visualisierungen für multivariate Daten geeignet sind. Dagegen widmet sich Abschnitt 3.3 univariaten Visualisierungsformen für das System, welche bereits in den Grundlagen behandelt wurden. Das Augenmerk liegt in diesem Abschnitt beim Verhalten der Visualisierung und dem möglichen Kontextverlust bei neu hinzukommenden Daten. Diese Evaluation wird später Einfluss bei der Wahl einer geeigneten Visualisierungsform haben.

Abschließend zeigt Abschnitt 3.4 verwandte Systeme zur Echtzeitvisualisierung von großen Datenmengen. In Anwendungen wie KNAVE-II und VISITORS wird die Bedeutsamkeit von Interaktionen und automatischen Analyseverfahren für Daten deutlich. MEDI+BOARD gibt erste Einblicke in die Gestaltung von Dashboards, deren generischem Verhalten und der Verlinkung von Unterelementen untereinander. Weiterhin zeigen Systeme wie die *Visual Analytics* Anwendung von Keim et. al und der LINE GRAPH EXPLORER kompakte Visualisierungen durch *Small Multiples*. Auch wird hier mit einer vertikalen Ausrichtung der *Line Graphs* und Pixel-orientierten Visualisierungen experimentiert. Diese Ansätze sollen auch in der späteren Konzeption verfolgt werden.



## 4. Anforderungsanalyse

In diesem Kapitel wird entsprechend dem bereits vorgestellten NESSEE System in Abschnitt 4.1 ein Anwendungsfall beschrieben. Dieser ermöglicht die Generierung von nicht-funktionalen (Abschnitt 4.2) und funktionalen (Abschnitt 4.3) Anforderungen an das zukünftige System zur Echtzeitvisualisierung von großen Datenmengen. Die NESSEE Anwendung soll als Konsument des neuen Systems betrachtet werden. Das zu entwerfende System wird im weiteren Verlauf auch als Service referiert, da es generische Funktionen auch für weitere Drittanbieter (Service-Konsumenten) bereitstellen soll.

### 4.1. Anwendungsfall: NESSEE

Das Projekt NESSEE wurde im Abschnitt 2.3 bereits vorgestellt. Testsysteme wie NESSEE enthalten eine Vielzahl infrastruktureller Daten. Es wäre von großem Nutzen diese Daten zu sammeln, sowie zur Laufzeit oder einem späteren Zeitpunkt auszuwerten. Das in dieser Arbeit zu entwerfende System zur Echtzeitvisualisierung von großen Datenmengen soll dies ermöglichen. Dafür sollen große Datensätze verarbeitet (Datenanalyse, Datenreduktion) und durch Visualisierungen sowie Interaktion dem Nutzer zugänglich gemacht werden.

Ein expliziter Anwendungsfall soll dabei helfen, die Anforderungen an das hier zu entwerfende System auszumachen.

Ein Testlauf in NESSEE soll eine Evaluation der Infrastruktur eines verteilten Systems durchführen. Dafür wird im NESSEE Projekt ein neuer *Test Case* mit vordefinierten Netzwerkparametern erstellt. Nachdem der *Test Case* über das Web-Interface von NESSEE ausgeführt wurde, wird die Infrastruktur des Systems entsprechend der gewählten Parameter des *Test Cases* emuliert. In dieser Zeit entstehen Log-Daten mit wichtigen Informationen zu Warnungen und Fehlern im System. Diese werden in einem *Central Log Repository* bereits von NESSEE gesammelt. Darüber hinaus generiert das emulierte System weitere Daten wie CPU-Last, Speicherverbrauch und I/O-Last. Diese Daten werden zur Zeit nicht weiter vom System verwertet und verworfen.

Dieser Umstand fordert einen Service, welcher sich der ungenutzten Daten annimmt, diese sammelt, verarbeitet und zur Analyse darstellt. Da die Daten über eine Zeitspanne des Testlaufs gesammelt werden, handelt es sich um zeitbasierte Daten. Üblicherweise sind diese mit einem Zeitstempel versehen.

Der Service soll eigenständig agieren und nicht direkt im NESSEE Projekt integriert werden. Schnittstellen sollen eine Verbindung zum Service zur Echtzeitvisualisierung von großen Datenmengen schaffen. Durch ein hohes Datenaufkommen müssen viele Daten in kurzer Zeit performant über eine Schnittstelle gesammelt und anschließend verarbeitet

## 4.2. Nicht-funktionale Anforderungen

werden. Zu einem bestimmten Zeitpunkt ist jedes System bei vielen Daten ausgelastet. Deshalb soll der neue Service Techniken zur Datenreduzierung bereitstellen. Weiterhin sollen die gesammelten Daten analysiert werden. Entsprechende Visualisierungen mit Interaktionstechniken zum Erkunden der zeitbasierten Daten sind daher als Ergänzung erforderlich. Abschließend sollen Daten vom Service auf statistische Merkmale analysiert werden. Die Ergebnisse werden dem Nutzer möglichst neben einer Hauptvisualisierung der Daten präsentiert.

Die Ermittlung der Anforderungen an das zu entwerfende System erfolgte weitestgehend mit Mitarbeitern des NESSEE Projekts. Demnach entstanden nicht-funktionale und funktionale Anforderungen, die im Weiteren betrachtet werden sollen.

### **4.2. Nicht-funktionale Anforderungen**

Dieser Abschnitt behandelt die nicht-funktionalen Anforderungen an den Service zur Echtzeitvisualisierung von großen Datenmengen. Die Anforderungsanalyse ergab nachfolgende Kriterien, welche vom Service möglichst erfüllt werden sollen.

Die Anforderungen erhalten für zukünftige Referenzierungen eine Nummerierung in Form von: NFyyyy. Y steht für eine fortlaufende Nummerierung.

**Unabhängigkeit** Das System soll eigenständig und unabhängig von Funktionalitäten anderer Anwendungen von Drittanbietern lauffähig sein. Als Service bietet es Schnittstellen für Service-Konsumenten an. Diese ermöglichen die Datenübergabe und Datenmanipulation. Als Konsequenz können mehrere Service-Konsumenten den Dienst zur Echtzeitvisualisierung von großen Datenmengen nutzen. [NF0010]

**Performanz** Die Datenstrukturen des Services sollen effektiv strukturiert und lose untereinander gekoppelt sein. Eine wirkungsvolle Komposition der Datenstrukturen gilt als eine wichtige Grundlage für ein performantes System und eine performante Datenübertragung durch den Service. [NF0020]

Weiterhin soll ein schmaler Kommunikationskanal eine schnelle Übertragung der Daten vom Service-Konsumenten zum Service ermöglichen. Demnach werden nur Daten übertragen, die tatsächlich vom Service für die auszuführende Funktion benötigt werden. [NF0030]

**Modularität** Nach dem *Separation of Concerns* Prinzip sollen eine Menge von Modulen im System verschiedene Verantwortlichkeiten in der Datenverarbeitung übernehmen.

Die Module sollen unabhängig voneinander agieren, austauschbar und erweiterbar sein. [NF0040]

### 4.3. Funktionale Anforderungen

Der folgende Abschnitt legt die funktionalen Anforderungen des Services fest. Die Anforderungsanalyse ergab nachfolgende Kriterien, welche vom Service möglichst erfüllt werden sollen.

Die Anforderungen erhalten für zukünftige Referenzierungen eine Nummerierung in Form von: xFyyy. Y steht für eine fortlaufende Nummerierung. X steht bei 0 für Muss-Kriterium, 1 für Kann-Kriterium mit einer weniger hohen Priorität und 2 für Abgrenzungskriterium.

**Modul zur Datenreduzierung** Ein Modul zur Datenreduzierung ermöglicht es, große Datenmengen nach ausgewählten Methoden permanent oder temporär auszudünnen. Eine permanente Ausdünnung der Daten bietet sich bei einer Überbelastung des Speichers durch zu große Datensätze an. Diese Ausdünnung muss zu einem festgelegten Zeitpunkt erfolgen. [0F0010]

Eine temporäre Reduzierung erachtet sich als sinnvoll bei dynamischen Datenvisualisierungen. Hier werden bei großen Datenmengen nur ein Bruchteil der Daten vom Service visualisiert. In einer Detailansicht können weitere Daten vom Service für eine detailliertere Visualisierung abgeholt werden. [1F0015]

Verschiedene Verfahren des *Samplings* von großen Datenmengen sind hier geeignet. [0F0020]

Weiterhin kann auch eine bestimmte Zeitspanne von Daten visualisiert werden. Die Daten innerhalb des Intervalls bleiben dabei vollständig erhalten. Methoden wie das *Dynamic Query Filter* sind hierfür erforderlich. [0F0030]

**Modul zur Datenanalyse** Ein Modul zur Datenanalyse ermöglicht es, Daten zu analysieren und die Ergebnisse dem Nutzer neben oder mit der Visualisierung zugänglich zu machen.

Die *Outlier Analysis* eignet sich für das Ausmachen von Anomalien in einer Zeitreihe. Durch die Analyse werden Ausreißer im Datensatz kenntlich gemacht und in der Visualisierung differenzierter dargestellt. [1F0040]

Daneben ermöglichen es Methoden zum *Forecasting* zukünftige Zeitwerte vorauszusagen. Dies hilft dem Nutzer mögliche Entscheidungen vorauszuplanen. [1F0050]

Neben den Hauptmethoden zur Analyse von Zeitreihen sind grundlegenden statistische

### 4.3. Funktionale Anforderungen

Methoden nicht unbedeutend. Hier sollen über mehrere Datenwerte beispielsweise Durchschnittswerte, Standardabweichungen und Varianz berechnet werden. [0F0060]

**Visualisierung** Die Visualisierung der Daten soll für den Nutzer eingängig aufbereitet werden. Vertraute Visualisierungsformen fördern dabei das Verständnis beim Nutzer. Auch die Wahl von Form und Farbe für Ausprägungen von Daten trägt dazu bei. Je nach Datenstruktur (univariate, multivariate Daten) stehen einfache und komplexere Visualisierungsformen zur Verfügung. Ein Zeitreihendatensatz sollte auf eine zeitbasierte Visualisierungsform abgebildet werden. Dafür kommen einige der behandelten Visualisierungsformen in Frage. Weiterhin soll der Kontextverlust bei neu hinzukommenden Daten in der Visualisierung möglichst gering gehalten werden. [0F0070]

Weitere mögliche Visualisierungen zur Stützung der Zeitreihenvisualisierung, zum ermöglichen von Interaktionstechniken oder zur Veranschaulichung der vom System analysierten Daten in abstrakter Form können hinzu genommen werden. [1F0080]

**Interaktionstechniken** Eine Menge von Interaktionstechniken sollen dem Nutzer dabei unterstützen die Visualisierung zu erkunden. Vorwiegend sei dabei das Mantra von Shneiderman [[Shneiderman, 1996](#)] mit den Schema *Overview*, *Zoom*, *Filter* und *Details-on-demand* zu beachten.

Das interaktive Filtern ist eine essentielle Interaktion. Indirekt wird damit auf einen Teil des Moduls zur Datenreduzierung zugegriffen. Dem Nutzer muss es so möglich sein, über eine Festlegung der Zeitspanne, den Datensatz auf das gewünschte Intervall zu reduzieren (*browsing*). [0F0090]

Weiterhin sollen durch interaktives Filtern Daten aufgrund ihrer Attribute reduzierbar gemacht werden (*querying*). Eine Art Legende in der Visualisierung soll das An- und Abwählen von Attributen ermöglichen. [0F0100]

Auch soll ein interaktives An- und Abwählen von Datensätzen es ermöglichen, nur ausgewählte Zeitreihen zu visualisieren. [0F0110]

Ein *Zooming* und *Panning* soll es dem Nutzer erlauben die Zeitreihen zu erkunden und besser zu verstehen. Üblicherweise wird die Visualisierung dafür in einen Overview-Bereich und eine Detail-Bereich aufgeteilt. Die Beschriftung der Zeitachse soll dabei einer zeitgerechten Granularität folgen. [0F0120]

In der herausgezoomten Ansicht besteht die Möglichkeit, die Daten reduziert darzustellen. Erst beim Hineinzoomen werden weitere Daten zur Verfügung gestellt, um dem Nutzer einen detaillierteren Einblick in die Daten zu gewähren. [1F0125]

Die interaktive Verzerrung bietet einen ähnlichen Ansatz wie das *Zooming* und *Panning*. Auch hier wird ein Detail-Bereich detaillierter dargestellt, während der Overview-Bereich

den restlichen Datensatz ausmacht. Da diese Interaktionstechnik der Intention des *Zooming* und *Panning* ähnlich ist, ist diese Art der Interaktion für das zu entwickelnde System nicht gewünscht. [2F0130]

Das *Linking* und *Brushing* ist essentiell für die Verknüpfung von Datenwerten. Die zu entwickelnde Visualisierung beinhaltet gemäß Anforderungsanalyse bisher mehrere Visualisierungsformen und Teilbereiche. Ein *Linking* und *Brushing* unterstützt den Nutzer bei der Zuordnung von Datenwertpaaren. [1F0140]

**Schnittstelle** Eine Schnittstelle für Service-Konsumenten ermöglicht die Datenübergabe und Datenmanipulation. Dafür müssen folgende Endpunkte zur Verfügung gestellt werden.

- Erstellung eines Datendepots [0F0170]
- Füllen des Datendepots mit Daten [0F0180]
- Löschen eines Datendepots [0F0190]

Weiterhin müssen innerhalb des Systems Schnittstellen zur Datenabfrage angeboten werden.

- Abfrage der Daten eines Datendepots [0F0200]

Das Abholen der Daten beinhaltet optionale Parameter zur Festlegung der *Dynamic Query Filter* Methode. [0F0210] Wie in der Anforderungsanalyse festgelegt, werden so Datensätze auf Zeitspannen oder Attribute eingegrenzt. Darüber hinaus wirkt sich dies im positiven Sinne auf die Übertragungsverzögerungen aus. Weiterhin soll bei der Erstellung eines Datendepots die permanente Datenreduzierung konfigurierbar sein. [0F0215]

**Echtzeit der Datenübertragung** Die Performanz des Systems ist ein bedeutsames Kriterium. Deshalb sollen folgende Zeiten für die Datenübertragung und Datenverarbeitung über die Schnittstelle und im System eingehalten werden (Tabelle 1).

Nummerierung	Beschreibung	Zeit
[0F0220]	Schreiben eines Datums in Datendepot	40 ms
[0F0230]	Abholen eines Datendepots mit 5000 Daten	500 ms

Tabelle 1: Anforderungen an die Datenübertragung

#### *4.3. Funktionale Anforderungen*

---

Nummerierung	Beschreibung	Zeit
[0F0240]	Darstellen eines Datendepots mit 5000 Daten	2000 ms
[0F0250]	Darstellen eines neuen Datums	500 ms

Tabelle 2: Anforderungen an die Datenvisualisierung

**Echtzeit der Datenvisualisierung** Neben der Datenübertragung soll auch die Darstellung schnellstmöglich erfolgen. Daher gilt es folgende Zeiten einzuhalten (Tabelle 2).

## 5. Konzepte zur eigenen Anwendung: Pipelinr

Im folgenden Kapitel wird das Gesamtkonzept der eigenen Anwendung mit dem Namen PIPELINR zur Visualisierung großer Datenmengen in Echtzeit entworfen. Die Architektur (Abschnitt 5.1) liefert hierbei einen Überblick der Anwendung und ihrer einzelnen Komponenten sowie Schnittstellen. Weiterhin werden das Backend (Abschnitt 5.2) und Frontend (Abschnitt 5.3) der Anwendung im Detail konzipiert.

### 5.1. Architektur

Ziel dieser Arbeit ist es, eine Lösung für das Problem der Echtzeitvisualisierung von großen Datenmengen zu entwickeln. Nach dem *Separation of Concerns* Prinzip soll es sich um eine verteilte Anwendung mit Backend und Frontend handeln. Während das Backend von PIPELINR für die Datenhaltung und Datenverwaltung zuständig ist, stellt das PIPELINR Frontend die Datenvisualisierung in Echtzeit bereit. Weiterhin wird eine eigenständige Anwendung gesucht, welche den Service zur Datenvisualisierung über Schnittstellen für Konsumenten anbietet. Abbildung 27 gibt einen zusammenfassenden Überblick der Architektur und Hauptfunktionen von PIPELINR.

Eine Fassade aus Schnittstellen im PIPELINR Backend bietet einen Endpunkt zur Datenbereitstellung und Manipulation für PIPELINR Konsumenten an. Darüber ist es möglich, neue Datenobjekte, sogenannte Pipelines, anzulegen. Pipelines können Datensätze beinhalten, welche wiederum eine Liste zeitbasierter Werte enthalten. Diese Pipelines werden später vom PIPELINR Frontend visualisiert. Der PIPELINR Konsument kann in den erstellten Pipelines neue Datensätze anlegen und diese jeweils in Echtzeit mit Werten aktualisieren. Weiterhin werden Teile der Schnittstelle auch zur internen Kommunikation zwischen PIPELINR Backend und Frontend genutzt.

Das PIPELINR Backend bietet darüber hinaus ein Modul zur Reduzierung großer Datenmengen an. In einer Datenbank werden neben Verwaltungsobjekten hauptsächlich die Pipelines und ihre Datensätze gespeichert. Daneben werden im Frontend verschiedene Verfahren zur Datenanalyse in den Visualisierungen angeboten.

Idealerweise bietet der PIPELINR Konsument im eigenen Frontend eine Verknüpfung zwischen eigenem Datenobjekt und der dazu angelegten Pipeline an. Dafür bietet sich die ID der Pipeline an, welche bei der Initialisierung zugeordnet wird. Dadurch ist es möglich, über einen Link vom Datenobjekt des Konsumenten direkt zur verknüpften Pipeline zu gelangen.

Ist eine bestimmte Pipeline in der Datenvisualisierung ausgewählt, werden die zu initialisierenden Daten vom PIPELINR Backend abgeholt, durch ein *Publish-Subscribe* Verfahren aktualisiert und im PIPELINR Frontend visualisiert und analysiert.

## 5.2. Backend

---

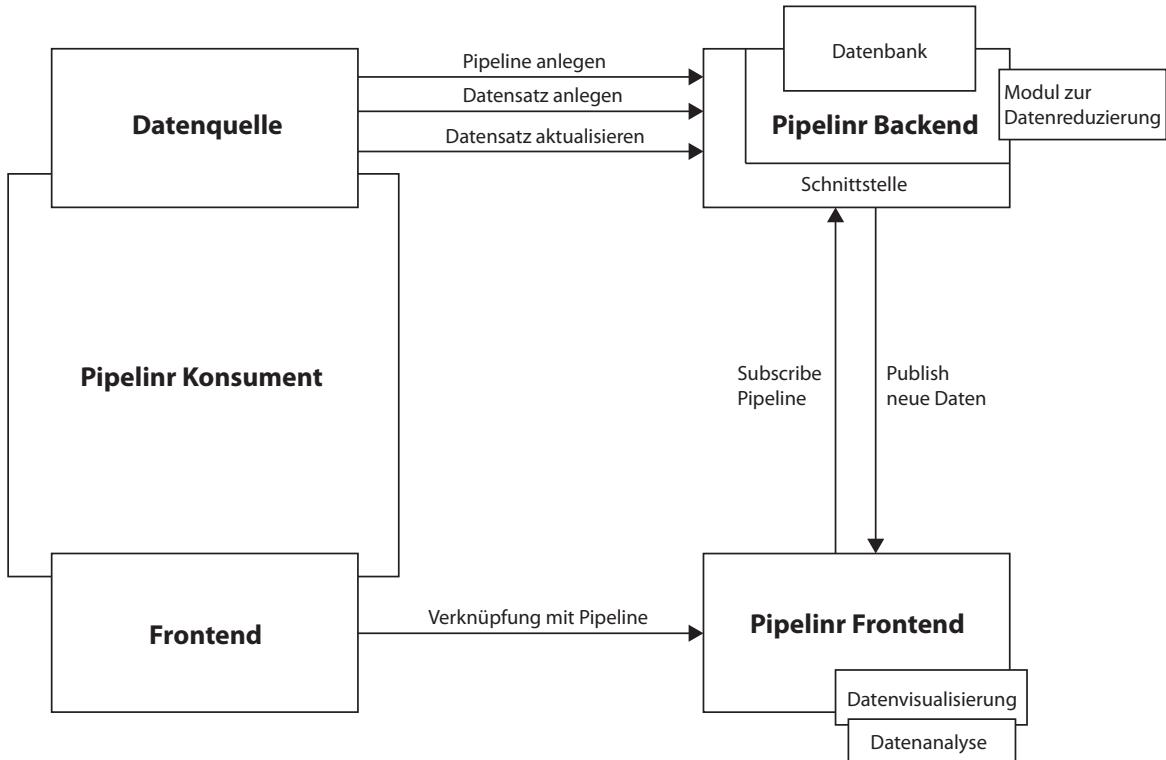


Abbildung 27: Überblick der Architektur von PIPELINR

## 5.2. Backend

In diesem Abschnitt wird das Backend von PIPELINR im Detail konzipiert. Dies beinhaltet die Zuständigkeiten der Datenverwaltung, Datenhaltung und Datenreduzierung. Über eine Schnittstelle zum PIPELINR Konsumenten werden stetig neue Daten bereit gestellt, welche über ein *Publish-Subscribe* Verfahren zum PIPELINR Frontend zur Visualisierung übermittelt werden.

### 5.2.1. Datenstruktur

Die Anforderungsanalyse erwartet eine lose Komposition der Datenstrukturen für eine performante Datenverwaltung und Kommunikation. Im Speziellen soll es möglich sein, ein Datenobjekt mit unterschiedlichen, in der Anzahl variierenden, Datensätzen anzulegen. Dies ermöglicht es, Datenobjekte mit mehreren Zeitreihen im späteren Verlauf zu

visualisieren. Das Datenobjekt wurde im vorangehenden Abschnitt als Pipeline vorgestellt.

Nutzer können für eine Pipeline einen Namen (`name`) und eine konfigurierbare Datenreduzierung (`reduction`) übergeben. Die Pipeline verwaltet eine Liste von Datensätzen (`Dataset`), die einem variierenden Typ (`type`) angehören. Der Typ wird in nominal und metrisch unterschieden. Diese Differenzierung wird benötigt, um in der späteren Visualisierung eine geeignete Visualisierungsform für nominale und metrische Werte zu wählen. Außerdem beinhaltet der Datensatz einen für die Pipeline eindeutigen Schlüssel (`key`) zur Unterscheidung der Datensätze. Weiterhin befinden sich die zu visualisierenden zeitbasierten Werte (`Value`) im Datensatz als Liste. Ein Datum muss aus einem nominalen oder metrischen Wert (`value`) und einem Zeitstempel (`timestamp`) bestehen. Optional kann das Datum eine Zusatzinformation (`enumeration`) beinhaltet. Diese kann in der späteren Visualisierung für eine weitere visuelle Kodierung verwendet werden. Einen Überblick der Datenstruktur liefert Abbildung 28.

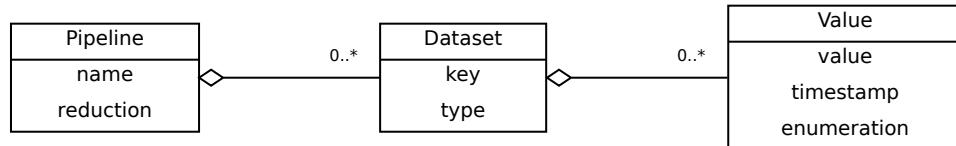


Abbildung 28: Datenstruktur einer Pipeline in PIPELINR

### 5.2.2. Fassade aus Schnittstellen

Die Anforderungsanalyse ergab diverse Schnittstellen, welche für die Datenverwaltung bereitzustellen sind. Die Schnittstellen können von PIPELINR Konsumenten und vom PIPELINR Frontend genutzt werden. Eine Fassade aus Schnittstellen eröffnet Endpunkte zum Bereitstellen und Manipulieren der Pipelines und deren Datensätze.

- Erstellung einer Pipeline
- Bearbeiten einer Pipeline
- Löschen einer Pipeline
- Abfrage aller Pipelines
- Abfrage einer Pipeline

## 5.2. Backend

---

Über die Schnittstelle können neue Pipelines mit einem Namen und einer konfigurierbaren Reduzierung angelegt werden. Weiterhin lassen sich je nach Abfrage alle oder individuelle Pipelines durch ihre ID abfragen. Optional besteht die Möglichkeit, weitere Parameter für die Abfrage einzelner Pipelines zur Datenreduzierung zu übermitteln. Beispielsweise würde durch das Setzen zweier Zeitstempel die angefragte Zeitspanne der Daten eingegrenzt werden. Abschließend soll auch das Löschen und die Bearbeitung einer Pipeline über die ID erfolgen (Liste 5.2.2).

- Erstellung eines Datensatzes
- Löschen eines Datensatzes
- Hinzufügen eines neuen Wertes im Datensatz

Neue Datensätze lassen sich mit einem, innerhalb der Pipeline, eindeutigen Schlüssel und einer Typenfestlegung (nominal oder metrisch) erstellen. Das Schreiben neuer Daten in einen Datensatz erfolgt über einem möglichst klein gehaltenes Datenvolumen für eine bestmögliche Performanz. Hierbei wird nur das Datum durch einen Zeitstempel, der Wert und optional die Zusatzinformation übertragen. Das Löschen eines Datensatzes erfolgt über die jeweilige ID (Liste 5.2.2).

### 5.2.3. Module und Kernfunktionen

Module sollen im PIPELINR Backend verschiedene Verantwortlichkeiten der Datenverarbeitung übernehmen. Diese werden in Form von Funktionen bereitgestellt. Mittels eines wohl eingesetzten *Separation of Concerns* Prinzips soll es in der Zukunft die Möglichkeit geben, neue Module hinzuzufügen und die Module durch Funktionen zu erweitern. Im Weiteren soll das erste Modul für die erste PIPELINR Version konzipiert werden.

**Modul zur Datenreduzierung** Ein Modul zur Datenreduzierung soll Funktionalitäten zur Ausdünnung der Daten bereitstellen. Die Reduzierung der Daten erfolgt über *Sampling* und *Dynamic Query Filter*. Im Weiteren werden die bereitzustellenden Funktionen erläutert.

*Dynamic Query Filter* beschreibt das Abrufen eines gefilterten Teilbereichs des Datensatzes. Das Modul soll dafür eine Funktion zur Festlegung einer Zeitspanne der Zeitserie bereitstellen (*browsing*). Die Funktionalität wird über zwei optionale Zeitstempel zur Eingrenzung des Datensatzes bei der Abfrage einer Pipeline ausgelöst. Die Schnittstelle liefert dann den Teilbereich der Datensätze einer Pipeline an das PIPELINR Frontend aus. Weiterhin beinhaltet *Dynamic Query Filter* das Filtern eines Datensatzes durch

Attribute (*querying*).

Das *Sampling* umfasst mehrere Methoden zur Ausdünnung der Daten. Einige davon wurden in den Grundlagen dieser Arbeit erläutert. Das Modul soll mindestens die Funktion des *Random samplings* zur zufälligen Ausdünnung und des *Stratified samplings* zur Ausdünnung nach der Dichte von Daten in einem Datensatz bereitstellen. Die Reduzierung durch das *Sampling* soll temporär oder permanent erfolgen.

Eine temporäre Reduzierung kann durch eine Abfrage einer Pipeline des PIPELINR Frontends erfolgen. Diese wird durch optionale Parameter, beispielsweise der Festlegung einer prozentualen Ausdünnung des *Random samplings*, bei der Abfrage einer Pipeline ausgelöst. Die neu entstandene Datenmenge wird danach durch die Schnittstelle zur Benutzung zurück zum PIPELINR Frontend überführt.

Eine permanente Reduzierung kann vom PIPELINR Backend für einen zu groß gewordenen Datensatz angestoßen werden. Für die Wahl des Zeitpunkts existieren zwei Möglichkeiten.

Die erste Möglichkeit beruht auf Ereignissen. Beim Hinzufügen eines neuen Wertes oder der Abfrage einer Pipeline, wird das Volumen der jeweiligen Datensätze auf einen Schwellwert überprüft. Bei der Überschreitung des Schwellwerts wird die Ausdünnung angestoßen. Der Nachteil besteht hierbei in der verschlechterten Performanz des Systems, da jede Schreib- oder Leseoperation zu einer Prüfung des Datensatzvolumens führt. Der Vorteil wäre die dauerhafte Überprüfung der Datensätze. So wäre sichergestellt, dass die Datensätze zwischen den Überprüfungen nicht zu groß werden.

Die zweite Möglichkeit besteht in dem Festlegen eines festen Zeitpunkts zur Datenreduzierung. Dies kann auch periodisch festgelegt werden, indem ein zeitliches Intervall für die Reduzierung definiert wird. Dadurch wird immer nach einer bestimmten Zeit das Volumen der Datensätze geprüft und gegebenenfalls das Datenvolumen reduziert. Nachteilig ist die unkalkulierbare Dynamik der verschiedenen Datensätze. Ein Datensatz kann beispielsweise zwischen den Intervallen plötzlich stark im Volumen wachsen.

Die zweite Möglichkeit soll der ersten Variante vorgezogen werden, da diese keine verschlechterte Performanz bei Lese- oder Schreiboperationen zulässt. Außerdem wird, wie bereits bei der Festlegung der Datenstruktur erwähnt, bei der Erstellung einer Pipeline durch eine konfigurierbare Datenreduzierung die Methode und weitere Parameter des periodischen *Samplings* festgelegt.

#### 5.2.4. Datenanalyse

In den Anforderungen wurde die Datenanalyse als eigenständiges Modul des Backends definiert. Jedoch besteht auch die Möglichkeit, diese Funktionalitäten in das Frontend zu verlagern. Der Nachteil bei einem Modul zur Datenanalyse im Backend wäre, dass

## 5.2. Backend

---

die Analyseergebnisse sich nur auf eine vollständig abgefragte Pipeline beziehen würden. Würde der Nutzer im Frontend zu Interaktionstechniken greifen, beispielsweise in die Datensätze hineinzoomen, müssten Analyseverfahren auf die Teildatensätze angewendet werden. Dies würde zu weiterer überflüssiger Kommunikation mit dem Backend bedeuten. Eine andere Möglichkeit wäre es, die Funktionen zur Analyse dem Frontend durch eine Schnittstelle zu übergeben. Dies würde jedoch zu viele Unkosten bedeuten, nur um einmalig eine Pipeline beim Abfragen im PIPELINR Backend zu analysieren, um danach bei Interaktionen weitere Analysen im Frontend stattfinden zu lassen. Einen Vorteil würde das Modul zur Datenanalyse im Backend erst bei zeitaufwendigen Analyseverfahren erzielen, weswegen sich für die Variante der Datenanalyse im PIPELINR Frontend entschieden wurde.

### 5.2.5. Erweiterte Kommunikation zum Frontend

Neben der bereits beschriebenen Schnittstelle des PIPELINR Backends, ergibt sich ein zusätzlicher Kommunikationskanal zwischen Backend und Frontend für die Echtzeitübertragung neuer Daten eines Datensatzes in einer Pipeline. Ein *Publish-Subscribe* Verfahren soll die Voraussetzung dafür schaffen.

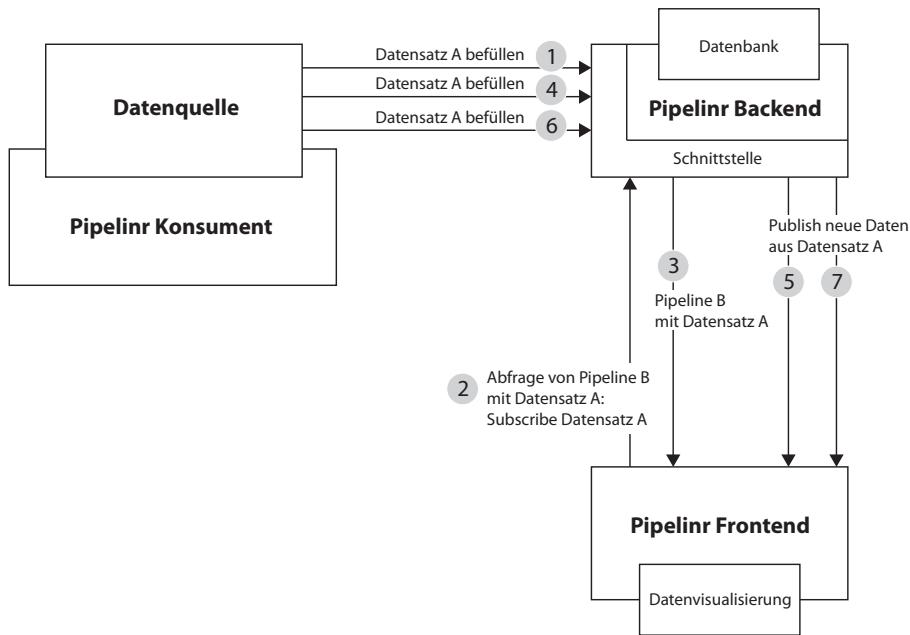


Abbildung 29: Kommunikationsablauf durch Publish-Subscribe Verfahren PIPELINR

Das initiale Abrufen einer Pipeline überträgt alle vorhandenen Datensätze. Dadurch wird im Weiteren die Möglichkeit geschaffen, dem Nutzer die Auswahl der zu visualisierenden Datensätze anzuvertrauen. Je nach Selektion werden die selektierten Datensätze für eine Echtzeitübertragung beim PIPELINR Backend registriert (*Subscribe*). Erfolgt ein Hinzufügen von neuen Daten durch PIPELINR Konsumenten in registrierten Datensätzen, werden die neuen Daten zum PIPELINR Frontend übertragen (*Publish*). Nach diesem Verfahren lassen sich neue Daten in kürzester Übertragungszeit im PIPELINR Frontend visualisieren. Abbildung 29 gibt einen Überblick des Ablaufs.

### 5.3. Frontend

In diesem Abschnitt wird das Frontend von PIPELINR im Detail konzipiert. Dies beinhaltet vor allem die Zuständigkeiten der Datenvisualisierung, Datenanalyse und Exploration von Daten durch Interaktionsmöglichkeiten. Interaktionen schaffen weiterhin die Voraussetzung zur Datenreduzierung über bereits behandelte Schnittstellen und Module aus dem PIPELINR Backend.

#### 5.3.1. Visualisierung

Die initiale Datenbeschaffung zur Visualisierung erfolgt über die bereits beschriebenen Schnittstellen des PIPELINR Backends. Neuzugänge von Daten in Datensätzen werden, falls diese Datensätze registriert wurden, mittels *Publish-Subscribe* zum PIPELINR Frontend übertragen. Weiterhin können Datensätze in einer Pipeline durch optionale Parameter in der Abfrage durch Datenreduzierung modifiziert werden.

In einem visuellen Analysesystem sind Designentscheidungen für die Visualisierung maßgebend. Der weitere Verlauf beschreibt die Motivation hinter Entscheidungen zur Konzeption der Visualisierung.

Die Visualisierung betrifft eine Pipeline und ihre Datensätze. Da es sich bei den Datensätzen um eine generische Anzahl von Zeitserien handelt, muss die Möglichkeit bestehen, generisch viele zeitbasierte Visualisierungen zu erzeugen. Die Darstellung aller Datensätze kann als Hauptvisualisierung betrachtet werden, während die visualisierten Anreicherungen aus Analyseauswertungen der Daten als Nebenvisualisierung gesehen werden kann.

**Ausrichtung der Visualisierungen** Die Visualisierung lässt sich horizontal oder vertikal ausrichten (Abbildung 30). Beide Herangehensweisen haben Auswirkungen auf Skalierbarkeit und Lesbarkeit bei einer Vielzahl von Datensätzen. Während es für Menschen aus westlichen Ländern natürlich ist, fortschreitende Zeitreihen von links nach rechts zu

### 5.3. Frontend

---

lesen, hat sich auch die *Timeline* Visualisierung (zu Lesen von oben nach unten) für textbasierte Ereignisse etabliert. Auch lassen sich bei einer vertikalen Ausrichtung neue Datensätze über den Bildschirmplatz hinaus durch *scroll-aware Content* einfügen. Das Scrollen in horizontaler Ausrichtung ist auch möglich, jedoch ungewohnt für den Nutzer.

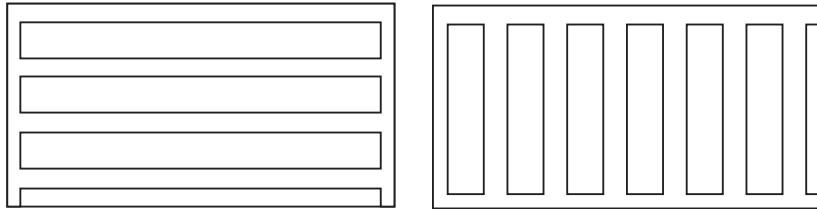


Abbildung 30: Horizontale und vertikale Ausrichtung von Datensätzen einer Pipeline in der Hauptvisualisierung

**Visualisierungsformen** Die Werte von Datensätzen der Hauptvisualisierung unterteilen sich in metrische und nominale Werte. Je nach Typ (`type`) des Datensatzes, muss die Zeitreihe anders visualisiert werden.

Die Wahl der Visualisierungsform für die metrischen Datensätze ist maßgebend für Faktoren zum Vergleich von Datensätzen sowie der platzsparenden Darstellung vieler Datensätze. Die Auswahl der Visualisierungsformen besteht aus den behandelten *Multi-Series Line Graph*, *Stacked Graphs*, *Horizon Graphs*, *Small Multiples* mit *Line Graphs* und Pixel-orientierten Visualisierungen. Da von einer generischen Anzahl von Datensätzen auszugehen ist, wird sich von *shared space* Visualisierungen wie *Multi-Series Line Graph* und *Stacked Graphs* zugunsten einer besseren Übersicht und eines Vergleichs der einzelnen Datensätze distanziert. Die Echtzeitübertragung birgt ein weiteres, bereits behandeltes Problem: Neu hinzukommende Datenpunkte, die außerhalb des Wertebereichs fallen, beeinflussen die Skalierung bei Achsen-basierten Visualisierungsformen wie *Line Graphs* und *Horizon Graphs*. *Horizon Graphs* müssen darüber hinaus in der Einteilung ihrer Bänder neu gezeichnet werden. In Pixel-orientierten Visualisierungen bewirken neue Datenpunkte außerhalb des Wertebereichs eine rechenintensive Neuberechnung aller Einfärbungen der Pixel. Alle Beobachtungen zusammengefasst, ergeben sich *Line Graphs* in Form von *Small Multiples* als bevorzugte Lösung für die metrischen Datensätze der Hauptvisualisierung.

Die nominalen Datensätze lassen sich durch eine eindimensionale Visualisierung abbilden. Die einzelnen Datenpunkte werden entlang einer Zeitleiste als Symbol eingetragen. Der Wert könnte beispielsweise später durch Interaktionstechniken wie *Details-on-*

*demand* abfragt werden. Aber auch die direkte textuelle Beschriftung des Wertes, vor allem in einer vertikalen Ausrichtung, wäre nachvollziehbar.

Abbildung 31 gibt einen abschließenden Überblick der Hauptvisualisierung in verschiedenen Ausrichtungen.

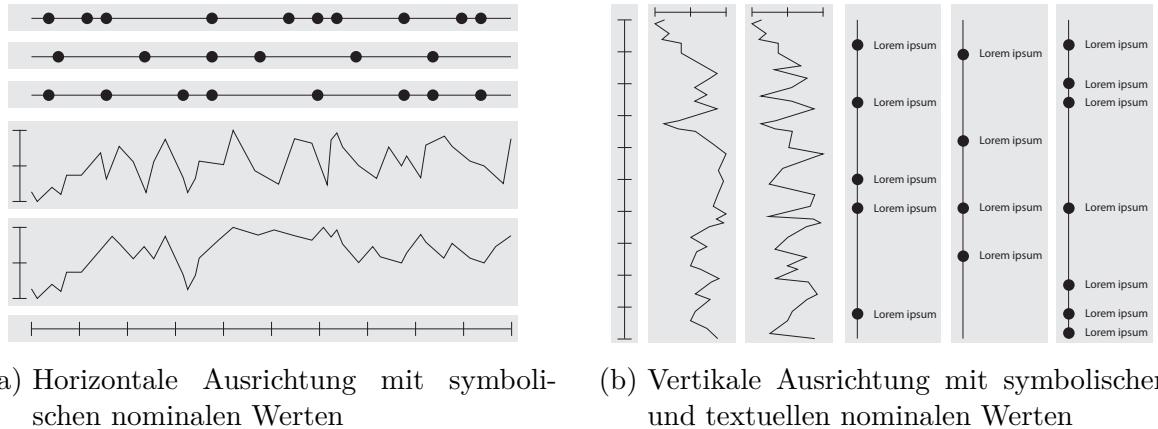


Abbildung 31: Hauptvisualisierung von drei nominalen und zwei metrischen Datensätzen

**Analysedarstellung** Die Analyse besteht aus statistischen Funktionen wie Varianz- oder Durchschnittswertberechnungen. Die vom Frontend bereitgestellten Methoden können auf metrische Datensätze angewendet werden. Da es sich um einzelne ausgerechnete Werte handelt, geschieht die Darstellung der Ergebnisse textuell. Für Berechnungen mit prozentualen Anteilen von Werten eignen sich auch Kreisdiagramme. *Bar Graphs* oder weitere *Line Graphs* für Analyseergebnisse sind wegen des Platzbedarfs vorerst als ungeeignet eingestuft. Die Darstellung der Analyseberechnungen erfolgt in horizontaler (Abbildung 32 (A)) oder vertikaler Ausrichtung zur Datensatzvisualisierung. Auch eine direkte Einzeichnung der Ergebnisse in den *Line Graphs* ist möglich.

### 5.3.2. Interaktionsmöglichkeiten

Interaktionen sind unerlässlich bei der Exploration von großen Datenmengen. Nutzer müssen eigenhändig einen Datensatz zum besseren Verständnis erkunden. Im weiteren Abschnitt werden Interaktionsmöglichkeiten für die Exploration der Daten in PIPELINR bedacht und konzipiert. Dabei erweitern einige Interaktionen die Visualisierung um Nebenvisualisierungen.

### 5.3. Frontend

---

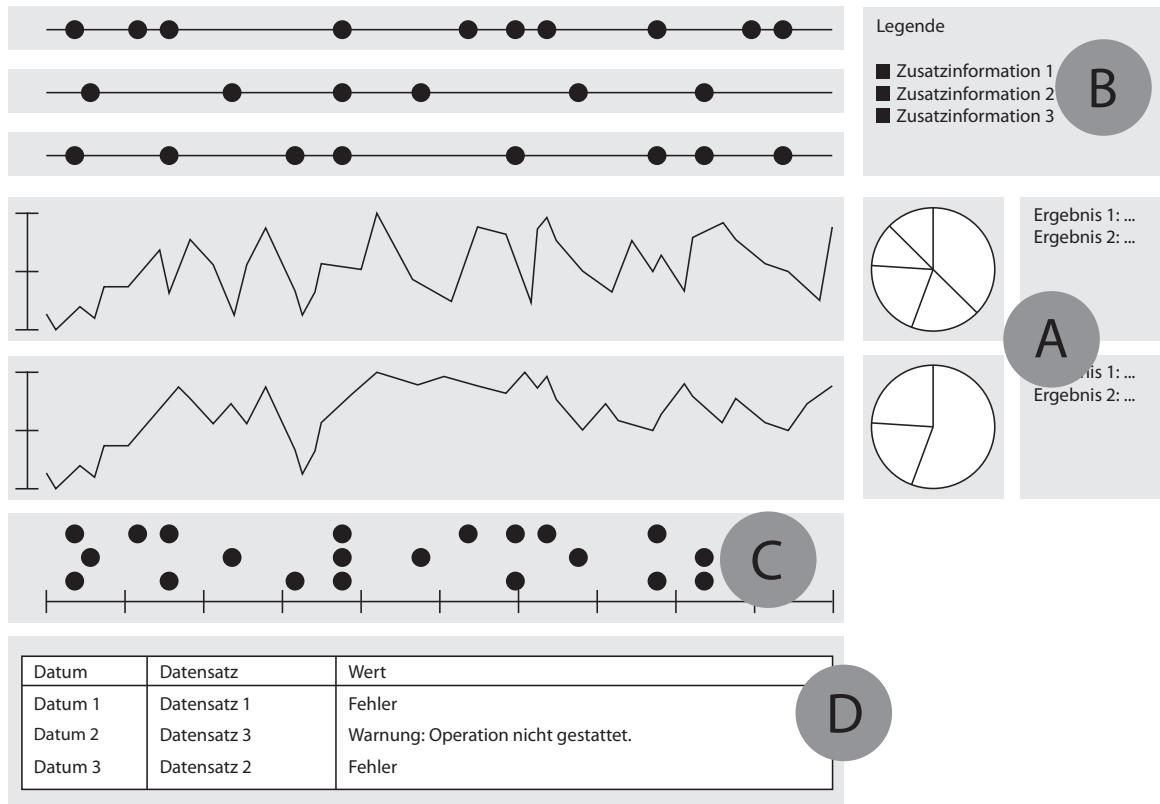


Abbildung 32: Horizontale Ausrichtung der Hauptvisualisierung mit Analysedarstellung (A), Legende (B), Zooming-Leiste (C) und Tabelle nominaler Werte (D)

**Legende** Eine Legende ist üblich zur Aufklärung von eingefärbten Bereichen einer Visualisierung. Die konzipierte Datenstruktur in PIPELINR erlaubt es, Zusatzinformationen (*enumeration*) bei Werten hinzuzufügen. Die Informationen sollen ein weiteres Attribut des Datums darstellen, welches nach einer neuen visuellen Kodierung verlangt. In die Visualisierung von PIPELINR soll dieses Attribut zur Einfärbung eines nominalen Datenpunkts verwendet werden (Abbildung 32 (B)).

Weiterhin soll die Legende eine Filterung im Frontend erlauben. Dadurch besteht die Möglichkeit, nominale Datenpunkte anhand ihrer Zusatzinformationen durch eine anklickbare Legende herauszufiltern. Es soll keine Kommunikation mit dem PIPELINR Backend für die Filterung der Zusatzinformationen nötig sein.

**Zooming und Panning** Das *Zooming* und *Panning* erlaubt es, große Datenmengen vereinfacht zu erkunden. Vor allem bei vielen nominalen Werten in einem Datensatz kommt es schnell zu Überschneidungen der visuellen Datenpunkte. Beim Hineinzoomen wird das Durcheinander der Datenpunkte aufgelöst.

Üblicherweise wird für ein *Zooming* und *Panning* die Visualisierung in einen Detail- und Overview-Bereich aufgeteilt. Der Detail-Bereich ist durch die Hauptvisualisierung bereits gegeben. Der Overview-Bereich (Abbildung 32 (C)) erweitert die Hauptvisualisierung und soll stets den Bezug zum gesamten Datenzeitraum vermitteln.

Weiterhin sollen mittels eines *Linking* nominale Datenpunkte aus den jeweiligen Detail-Bereichen im Overview-Bereich dupliziert dargestellt werden. Ein *Brushing* soll es erlauben, Teilbereiche im Overview zu markieren (*Zooming*) und zu verschieben (*Panning*). Diese Manipulation hat Auswirkungen auf den Detail-Bereich. Ein Verschieben des markierten Teilbereichs ist ein simultanes *Panning* in den verknüpften Detail-Bereichen. Ebenso bedeutet ein Verkleinern des markierten Teilbereichs ein *Zooming* in die Detail-Bereiche. Abbildung 33 gibt einen ersten Einblick in das Verhalten der Visualisierung bei Interaktion.

Der Overview-Bereich könnte neben den nominalen Datensätzen auch die metrischen Datensätze zusammenführen. Dies stellt sich aus folgenden Gründen als negativ heraus. Ein erstes Problem ist der variierende Wertebereich metrischer Datensätze. Die Werte müssten erst normalisiert werden. Danach würden diese nur noch ein verzerrtes Bild der einzelnen Datensätze wiedergeben. Weiterhin würden zu viele *Line Graphs* den Overview-Bereich unübersichtlich machen. Die nominalen Werte hingegen wurden bisher nur eindimensional in den jeweiligen Detail-Bereichen dargestellt. Dies birgt den Vorteil, eine weitere Dimension für die Anordnung der verschiedenen nominalen Datensätze zu verwenden. Die Datenpunkte würden dann übereinander dargestellt werden (Abbildung 32 (C)).

**Linking und Brushing** Neben dem verwendeten *Linking* und *Brushing* für den Overview- und Detail-Bereich, sollen weitere Visualisierungen miteinander verknüpft werden.

Die Darstellungen der Analyseergebnisse in Form von Kreisdiagrammen lassen sich mit den Visualisierungen der zugehörigen Datensätze verknüpfen. Ein *Zooming* in den Detail-Bereichen passt die Analysedarstellungen für den Teildatensatz an.

Weiterhin soll eine nach Datum sortierte Liste aus Werten aller nominalen Daten existieren (Abbildung 32 (D)). Ein *Brushing* einer zusammenhängenden Abfolge von Listen-Einträgen führt zu einem *Zooming* in die Detail-Bereiche für den festgelegten Zeitbereich. Abbildung 33 verdeutlicht noch einmal die Auswirkungen der verknüpften Visualisierungen durch das *Linking* und *Brushing*.

### 5.3. Frontend

---



Abbildung 33: Das Markieren eines Teilbereichs im Overview-Bereich (blau) beeinflusst die anderen Visualisierungen. Die Detail-Bereiche werden auf den markierten Bereich hineingezoomt (grün). Weiterhin sind zwei nominale Werte in der Markierung betroffen, welche in der Liste aller nominalen Werte markiert werden (rot). Die Analysedarstellung passt ihre Berechnungen auf den eingeschränkten Bereich der Detail-Bereiche an (gelb).

**Datenreduzierung** *Dynamic Query Filter* ermöglicht es, die Datensätze in einer Zeitspanne zu begrenzen. Zwei optionale Parameter in Form von Zeitstempeln legen die Zeitspanne in der Abfrage einer Pipeline fest. Diese lassen sich vom Nutzer in einem Formular eingeben, bevor eine Pipeline vom PIPELINR Backend angefordert wird. Weiterhin soll die Möglichkeit bestehen, über Auswahlfelder nur ausgewählte Datensätze einer Pipeline anzeigen und für Aktualisierungen registrieren zu lassen. Auch kann die temporäre Datenreduzierung aus dem PIPELINR Frontend heraus erfolgen. Im Formular zur Abfrage einer Pipeline sollen dazu Eingabefelder angeboten werden.

### 5.3.3. Verknüpfung mit Pipelinr Konsumenten

Die Verknüpfung des Datenobjekts vom PIPELINR Konsumenten und der Pipeline erfolgt über eine bei der Erstellung generierte ID der Pipeline. Die ID ermöglicht eine generische Navigation mittels eines Permalinks zur ausgewählten Pipeline in der PIPELINR Anwendung. Die [URL](#) kann vom PIPELINR Konsumenten in der eigenen Anwendung zur direkten Verlinkung mit der Visualisierung verwendet werden.



## 6. Implementierung

PIPELINR ist eine Webanwendung basierend auf den Web-Technologien JavaScript, CSS und HTML. Daraus entsteht der Vorteil von Plattform- und Ortsunabhängigkeit. In diesem Kapitel wird die Implementierung der Anwendung vorgestellt. Hierbei werden verwendete Frameworks aufgezeigt, umgesetzte Teilbereiche aus dem Konzept dargelegt und Implementierungsdetails für komplexere Komponenten erläutert. Auch werden Entscheidungen zu nicht umgesetzten Konzepten dargelegt. Das Kapitel wird im Weiteren in die zwei Abschnitte Backend [6.1](#) und Frontend [6.2](#) aufgeteilt.

### 6.1. Backend

Das PIPELINR Backend verwendet NODE.JS. Das Framework wurde 2009 von Ryan Dahl ins Leben gerufen und ermöglicht die Nutzung von serverseitigen JavaScript auf einem Applikationsserver. Dadurch wird die Möglichkeit geschaffen, JavaScript gleichzeitig im Frontend sowie auch im Backend einer Webanwendung zu nutzen. Anwendungen, die im Frontend sowie Backend JavaScript verwenden, werden auch als *Full-Stack* JavaScript Anwendungen bezeichnet.

Viele Eigenschaften qualifizieren NODE.JS zur Nutzung für PIPELINR. NODE.JS verwendet mit der GOOGLE V8 einen der schnellsten JavaScript-Compiler. Dieses Merkmal stellt eine wichtige Basis für hochperformante und parallelisierbare Anwendungen dar. Auch ist die Unterstützung von Echtzeitübertragungen durch Streaming und Websockets eines der Grundkonzepte in NODE.JS zur Auslieferung neuer Daten an Clients. Weiterhin ist Node Packaged Modules ([npm](#)) der offizielle *package manager* von NODE.JS. Dieser bietet die Möglichkeit, eine Anwendung durch neue Module des NODE.JS-Ökosystems zu erweitern. Abschließend ist die Nutzung von *Full-Stack* JavaScript ein bedeutsamer Ansatz zur Wiederverwendung von Programmiercode auf Client- sowie Serverseite.

Skalierung ist ein essentieller Teil von NODE.JS. Oft begegnen Applikationen dem Problem, eine hohe Anzahl an Clients bedienen zu müssen. In klassischen Webservern wird hierfür je eingehende Abfrage ein eigener Thread bereitgestellt. Das Problem besteht hier in der niedrigen Skalierbarkeit von Threads. *“Wird für jeden Thread ein Stack von lediglich zwei MByte reserviert [...], erfordern 10.000 gleichzeitig stattfindende Zugriffe bereits 20 GByte Speicher.”* [Rodden, 2014] Weiterhin stoßen auch Betriebssysteme bei einer hohen Anzahl von Threads an ihre Grenzen. NODE.JS begegnet diesem Problem auf anderer Art und Weise. Hier arbeitet ein einzelner Thread die Abfragen nacheinander ab. Ein ausgelagertes und asynchrones I/O-System lässt das System auch bei einer hohen Anzahl von Abfragen nicht blockieren. [Rodden, 2014]

*Node.js* ist abschließend betrachtet eine außerordentlich geeignete Wahl für skalierbare,

## 6.1. Backend

---

hochperformante und echtzeitfähigen Applikationen.

### 6.1.1. Schnittstellen

Die im Konzept erarbeiteten Schnittstellen wurden durch das Framework EXPRESS umgesetzt. Neben erweiterten Konfigurationen des Webservers, ermöglicht das Framework die Implementierung des Routings für eingehende Abfragen an das Backend. Demnach ist es möglich, einen kompletten Webservice durch EXPRESS zu implementieren. Tabelle 3 veranschaulicht die Schnittstellen für Pipelines, Datensätze und Werte. Die Abfrage einer Pipeline kann durch optionale Parameter zur Datenreduzierung angereichert werden. Eine detaillierte Schnittstellenbeschreibung für Service-Konsumenten ist im Anhang A.1 dieser Arbeit zu finden.

Auch soll es bei der Erstellung einer Pipeline möglich sein, das Verhalten der vom Backend angestoßenen periodischen Datenreduzierung festzulegen. Dies kann die Art der Reduzierung betreffen, aber auch Parameter wie beispielsweise die Rate des *Samplings*. Bei keiner Festlegung werden Standardwerte gewählt. Auch hierfür können detailliertere Beschreibungen im Anhang A.1 ausgemacht werden.

Darüber hinaus existieren weitere Schnittstellen zum Registrieren und An- beziehungsweise Abmelden eines Nutzers. Die Implementierung der Schnittstellen erfolgte nach dem Representational State Transfer ([REST](#)) Paradigma.

Operation	Ressource	Beschreibung
POST	..../pipelines	Erstellung einer Pipeline
GET	..../pipelines	Abfrage aller Pipelines
GET	..../pipelines/:id	Abfrage einer Pipeline
DELETE	..../Pipelines/:id	Löschen einer Pipeline
PUT	..../Pipelines/:id	Bearbeiten einer Pipeline
POST	..../pipelines/:id/datasets	Erstellung eines Datensatzes
DELETE	..../pipelines/:id/datasets/:id	Löschen eines Datensatzes
PUT	..../pipelines/:id/datasets/:id	Aktualisieren eines Datensatzes
POST	..../pipelines/:id/datasets/:id/values	Erstellung eines Wertes

Tabelle 3: Schnittstelle zum Backend für Pipelines, Datensätze und Werte

### 6.1.2. Modularität

Eine weitere Anforderung bestand darin Modularität zu gewährleisten. Unter anderem soll es so möglich sein, ein Modul zur Datenreduzierung bereitzustellen. Auch weitere

Module können so im späteren Verlauf hinzugefügt werden.

Das Modulsystem von NODE.JS bietet hierfür eine elegante Lösung. Funktionen aus einer losen JavaScript-Datei können in einem Objekt namens `module.exports` definiert werden. Durch das Modulsystem werden die exportierten Objekte aus einer Datei dann über `require('filename.js')` nach außen verfügbar gemacht.

**Modul zur Datenreduzierung** Nach dem voran gegangen Prinzip zur Modularisierung wurde das Modul zur Datenreduzierung implementiert. Dieses beinhaltet folgende Funktionen:

- Auswahl der Datensätze einer Pipeline
- Begrenzen der Datensätze durch eine Zeitspanne
- Temporäres Ausdünnen der Datensätze durch *Sampling* Methoden
- Permanentes Ausdünnen der Datensätze durch *Sampling* Methoden

Die *Sampling* Methoden beinhalten eine Auswahl aus *Random sampling*, *Stratified sampling* und einem *Sampling* nach Intervallen.

#### Beispiel 6.1 Intervall Sampling

Das *Sampling* nach einem Intervall ermöglicht es Datensätze in bestimmten Abständen auszudünnen. Demnach bedeutet eine Rate des *Samplings* von 5, dass jeder 5. Wert behalten wird, während die restlichen Werte ausgedünnt werden.

Weiterhin können die Funktionen zur Datenreduzierung in der vorliegenden Datei erweitert werden. Dadurch besteht die Möglichkeit, beispielsweise neue Funktionen zur Datenreduzierung oder speziellere Methoden zum *Sampling* hinzuzufügen.

### 6.1.3. Prozess der Datenreduzierung

Über die zuvor definierte Schnittstellen kann die *Abfrage einer Pipeline* erfolgen. Die Abfrage kann, wie beschrieben, durch optionale Parameter zur Datenreduzierung angereichert werden. Beispielsweise werden für das *Begrenzen der Datensätze durch eine Zeitspanne* der Funktionsname `trimPipeline` mit den Parametern `begin` und `end` übergeben. Die übergebenen Funktionen und ihre Parameter werden dann vom Backend generisch aufgelöst und an die verfügbaren Module mitsamt ihrer Funktionen übergeben.

## 6.1. Backend

---

Durch die gegebene Erweiterbarkeit können auch weitere Funktionen über die Datenreduzierung hinaus durch Module angeboten werden. In der *Abfrage einer Pipeline* müssen dann lediglich der definierte Funktionsname und die benötigten Parameter mitgesendet werden.

Eine weitere Datenreduzierung wird periodisch vom Backend angestoßen. Hierfür wurde die Bibliothek AGENDA verwendet. AGENDA ermöglicht es, in Anwendungen sogenannte Jobs zu definieren. Die Jobs können dann periodisch oder zu einem gewünschten Termin angestoßen werden. In PIPELINR wird ein Job zur permanenten Ausdünnung von Daten registriert. Dieser greift auf das Modul zur Datenreduzierung zu, um die Datensätze nach dem überschreiten einer Größe nach einem gewählten *Sampling* auszudünnen.

### 6.1.4. Datenspeicherung

Gegenüber den Mitbewerbern REDIS und CASSANDRA (verwandte Arbeiten 3.1), fiel die Wahl zur Datenspeicherung auf die NoSQL Datenbank MONGODB. REDIS setzt auf Schlüssel-Wert-Daten mit einer hohen Performanz in der Speicherung durch Nutzung des Arbeitsspeichers. Die minimalistische Datenstruktur und das geringe Speichervolumen entsprach jedoch nicht dem Anwendungsfall von PIPELINR. CASSANDRA hingegen birgt großes Potenzial in der Skalierbarkeit und Ausfallsicherheit. In einer wachsenden PIPELINR Anwendung mit Analyseverfahren im Backend, könnte CASSANDRA in Zukunft eine größere Rolle einnehmen.

Die Schnittstelle zwischen Backend und MONGODB wird durch die Bibliothek MONGOOSE vereinfacht. MONGOOSE ist für das Object Data Mapping ([ODM](#)) verantwortlich, ähnlich dem Object Relational Mapping ([ORM](#)), und übersetzt demnach die Dokumente der Datenbank in JavaScript Objekte und umgekehrt. Dadurch können *Schemes* der konzipierten Datenobjekte (Pipeline, Datensatz, Wert) erstellt werden und als *Models* zur Interaktion mit der Datenbank verwendet werden.

In der Datenstruktur wurde sich darauf geeinigt, neben einer generischen Anzahl von metrischen Datensätzen, nur einen nominalen Datensatz zu verwenden.

### 6.1.5. Websockets

Ein weiterer Aspekt von PIPELINR ist die Echtzeitfähigkeit. Clients sollten über Änderungen mittels eines *Publish-Subscribe* Verfahrens benachrichtigt werden. Vor allem für neue Werte in einem Datensatz ist dies essentiell für die Echtzeitvisualisierung der Daten. Daher wurde die Bibliothek SOCKET.IO zur Verwendung von Websockets benutzt. Dies erlaubt es, vom Backend Daten an alle Clients zu schicken, welche sich für ein bestimmtes Ereignis registriert (*Subscribe*) haben. Der folgende Programmcode zeigt

das Versenden (*Publish*) von Werten zu allen registrierten Clients des Ereignisses des jeweiligen Datensatzes.

```
io.sockets.emit('add_value_' + value.datasetId, { value: value });
```

Durch die Nutzung von MONGOOSE zur Datenspeicherung ergeben sich weitere Funktionalitäten. Die Middleware hält Events wie `pre` und `post` für die zuvor definierten *Schemas* zur Datenspeicherung bereit, welche beispielsweise, wie im folgenden Beispiel, vor der Speicherung neuer Werte ausgelöst werden. Dadurch kann die Middleware genutzt werden, um registrierte Clients bei Datenänderungen zu benachrichtigen. Datenspeicherung und Websockets sind somit eng miteinander verwoben.

```
valueSchema.post('save', function (value) {
  io.sockets.emit('add_value_' + value.datasetId, { value: value });
});
```

## 6.2. Frontend

Das PIPELINR Frontend verwendet ANGULAR.JS von GOOGLE. Es ermöglicht für große Webanwendungen eine klare Strukturierung durch das MVC-Muster. Dabei übernehmen *Controller* die Steuerung zwischen *Model* und *View*, die *Views* werden als HTML-Dateien definiert und das *Model* ist die Schnittstelle zum Backend. Eine bidirektionale Datenbindung ermöglicht die Aktualisierung der *Views* nach einer Änderung des *Models*.

Daneben existieren weitere Elemente in ANGULAR.JS wie *Services* und *Directives*. In *Services* wird die Geschäftslogik untergebracht. Dadurch wird der Programmiercode im *Controller* möglichst kurz gehalten und die Funktionalitäten sind in mehreren Bereichen aus dem *Service* heraus wiederverwendbar. Außerdem lässt sich in den *Services* die Kommunikation zum Backend über REST und Websockets umsetzen.

Daneben ermöglichen es *Directives* neue HTML-Elemente zu erstellen. Einmal eine *Directive* definiert, kann diese in allen Bereichen der Anwendung als HTML-Element wiederverwendet werden. In einem späteren Abschnitt wird auf die Nutzung von *Directives* für wiederverwendbare HTML-Elemente zur Visualisierung in PIPELINR näher eingegangen.

### 6.2.1. Anwendungsbereiche

Die Anwendung teilt sich in die zwei Hauptbereiche Überblick der Pipelines und Detailansicht einer Pipeline auf.

## 6.2. Frontend

---

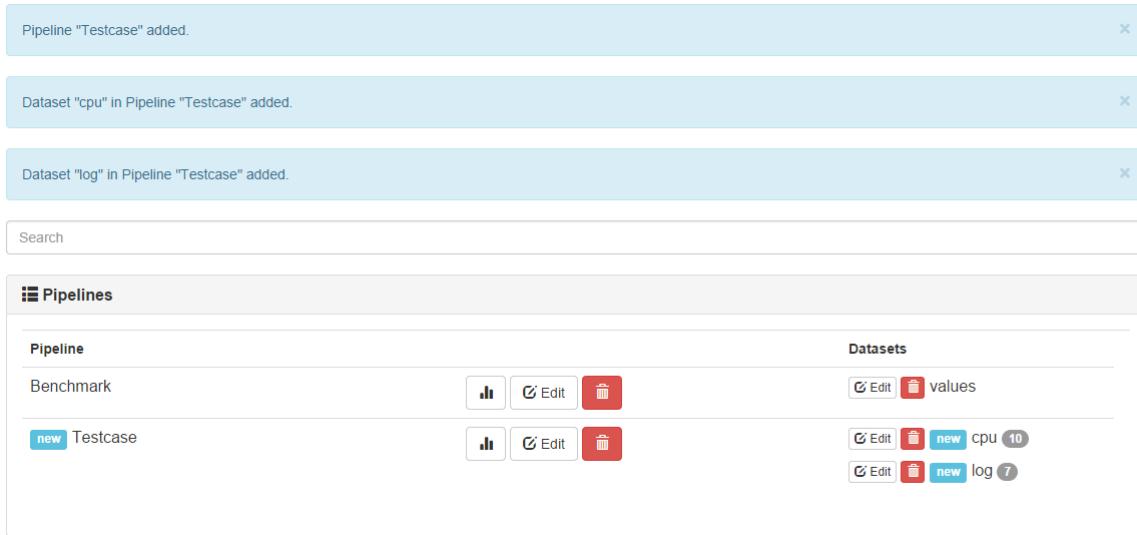


Abbildung 34: Überblick der Pipelines in Pipelinr

Im ersten Bereich erhält der Nutzer einen Überblick aller Pipelines (Abbildung 34). Die einzelnen Pipelines können editiert, gelöscht oder angeschaut werden. Letzteres führt zur Detailansicht einer Pipeline. Weiterhin werden für jede Pipeline die Datensätze dargestellt. Auch hier besteht die Möglichkeit zum Editieren oder zur Löschung. Durch die Registrierung von Websockets bekommt der Nutzer neue Aktualisierungen über Löschen und Erstellungen von Pipelines und Datensätzen in Form von Mitteilungen und Darstellung neuer Pipelines angezeigt. Auch neue Werte in einem Datensatz werden durch einen Zähler dargestellt.

Der zweite Bereich ist die Detailansicht einer Pipeline. Ein Dashboard visualisiert alle Details zu einer Pipeline 36 und ein Verarbeitungsfenster 35 ermöglicht es, die Pipeline modifiziert durch Datenreduzierung vom Backend abzuholen.

**Verarbeitungsfenster** Das Abholen einer modifizierten Pipeline erfolgt über die Eingabe der bereits beschriebenen optionalen Parameter. Dafür wird im Frontend eine Funktionssammlung erstellt, mit Funktionsnamen und optionalen Parametern, welche dann generisch vom Backend in den Modulen aufgelöst werden kann, falls diese bei der Abfrage mitgesendet wird. Sobald ein neues Modul oder eine neue Funktion im Backend hinzugefügt wird, kann im Frontend das Verarbeitungsfenster erweitert werden, um weitere optionale Parameter für die neue Funktion zu übergeben. Die Auflösung der Funktionen im Backend erfolgt generisch.

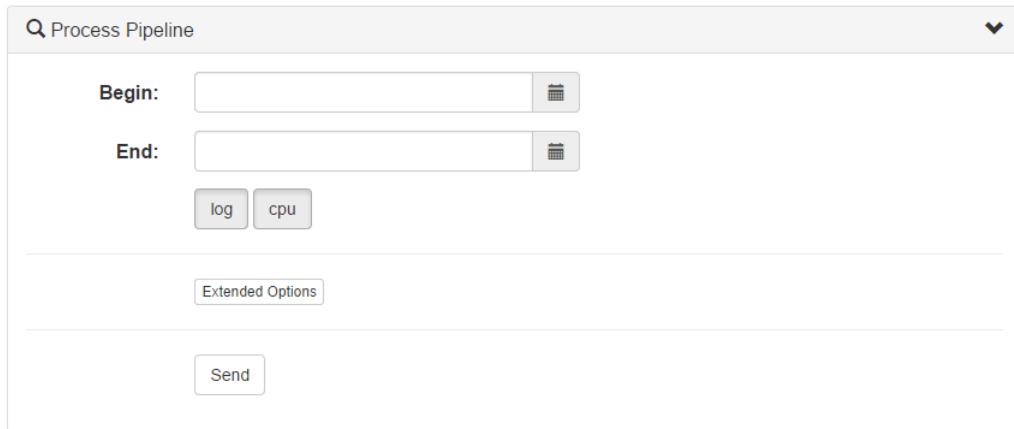


Abbildung 35: Abfrage einer Pipeline in Pipelinr

Bisher lassen sich Methoden zur Datenreduzierung über das Verarbeitungsfenster auslösen. Die Datensätze können dadurch selektiert, in einer Zeitspanne begrenzt und durch *Sampling* ausgedünnt werden. Letztere Methode befindet sich unter einem ausklappbaren *Extended Options* Menü. Das Selektieren der zu visualisierenden Datensätze hat gleichzeitig den Effekt, dass sich der Client nur für diese Datensätze der Pipeline für Aktualisierungen registriert. Auf den Kommunikationskanal vom Frontend zur Echtzeitübertragung wird später noch einmal genauer eingegangen.

**Dashboard** Das Dashboard ist zuständig für die Visualisierung einer Pipeline und der zugehörigen Datensätze. Gemäß des Konzepts wurden zwei zeitbasierte Visualisierungen, *Line Graph* und *Point Graph*, in Form von *Small Multiples* für die metrischen und nominalen Datensätze implementiert. Die Ausrichtung erfolgte zugunsten einer gewohnten Lesbarkeit und vertikalen Skalierbarkeit in vertikaler Form. Die Visualisierung der nominalen Daten ermöglicht es, *Details-on-demand* abzufragen.

Weiterhin existiert eine Legende zur Filterung und eine tabellarische Übersicht der nominalen Werte. Eine Zooming-Leiste (*Overview*) ermöglicht das *Zooming* und *Panning* für alle Datensätze. Auch fasst dieser Bereich alle nominalen Werte zusammen (Abbildung 36).

Das *Zooming* und *Panning* in der Zooming-Leiste ist eine der umgesetzten Interaktionsmöglichkeiten. Dadurch lassen sich die Bereiche in den Visualisierungen des nominalen und der metrischen Datensätze eingrenzen. Weiterhin beeinflusst die Filterung in der Legende alle Visualisierungen mit nominalen Datensätzen.

Wie bereits erwähnt, besteht die Möglichkeit neue HTML-Elemente über ANGULAR.JS

## 6.2. Frontend

---



Abbildung 36: Dashboard in Pipelinr

*Directives* zu deklarieren. Das Dashboard besteht aus mehreren einzelnen *Directives*. Die Visualisierung der metrischen Daten erfolgt beispielsweise über eine solche *Directive*. Wenn eine Pipeline mehrere metrische Datensätze aufweist, werden die einzelnen Visualisierungen über eine einzelne *Directive* durch Iteration über die Datensätze erstellt. Weiterhin wurde die Analysefähigkeit der metrischen Daten, welche zuerst im PIPELINR Backend sein sollte, in die *Directives* ausgelagert. So ist es möglich, statistische Daten im *Line Graph* darzustellen. Auch die Legende, die zeitbasierte Visualisierung eines nominalen Datensatzes und die Tabelle der nominalen Datensätze sind *Directives* und können in

anderen Bereichen der Anwendung wiederverwendet werden. Die Visualisierungen selbst wurden mit dem Framework D3.js umgesetzt. Frameworks wie D3.js können als Modul in ANGULAR.js geladen, um dann beispielsweise in *Directives* verwendet zu werden.

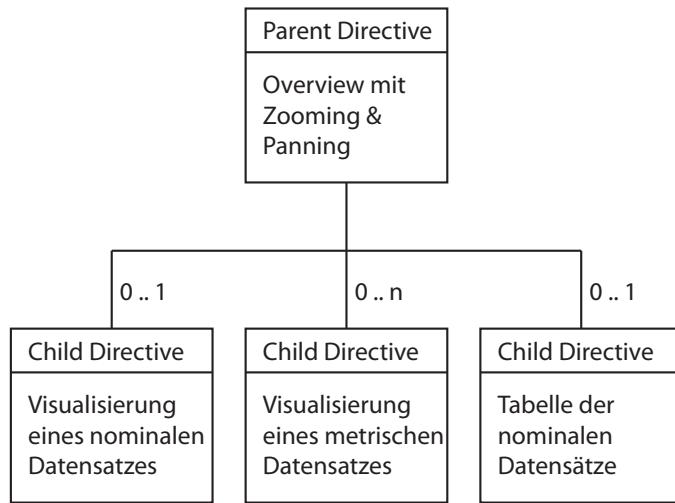


Abbildung 37: Aufbau der Directives Struktur

Eine Problematik, die sich aus den vielen unabhängigen *Directives* ergibt, ist die fehlende Interaktion untereinander. Die Zooming-Leiste (Overview) soll das *Zooming* und *Panning* ermöglichen. Dafür muss die Möglichkeit geschaffen werden, auf die anderen *Directives* zuzugreifen, um diese bei Interaktion zu verändern. Dies führte zum Ansatz der verschachtelten *Directives* in ANGULAR.js. Während die Zooming-Leiste eine *Parent Directive* ist, beinhaltet diese nun die zuvor deklarierten HTML-Elemente als *Child Directives*. Dadurch kann beim *Zooming* und *Panning* immer auf die *Child Directives* zugegriffen werden (Abbildung 37).

### 6.2.2. Echtzeitvisualisierung

In der Detailansicht einer Pipeline registriert sich der Client für Aktualisierungen aller Datensätze beim PIPELINR Backend. Durch das Verarbeitungsfenster lassen sich nicht nur die zu visualisierenden Datensätze angeben, sondern auch diejenigen Datensätze, die für Aktualisierungen beim Backend registriert werden sollen.

Die *Parent Directive* im Dashboard hat sogenannte *Watcher* für Aktualisierungen der Pipeline. Bei einem neuen Wert in einem Datensatz werden alle *Child Directives*, welche

## *6.2. Frontend*

---

diesen neuen Wert darstellen sollen, benachrichtigt. Während für einen nominalen Wert die zeitbasierte Visualisierung der nominalen Werte, die Werte in der Zooming-Leiste und in der Tabelle aktualisiert werden, ist bei einer Aktualisierung eines metrischen Wertes nur die jeweilige zeitbasierte Visualisierung betroffen. Unabhängig von der Art des Wertes wird die Zeitspanne für alle X-Achsen aktualisiert.

## 7. Evaluation

In der Evaluation wird das Gesamtsystem auf die zuvor gestellten Anforderungen (Kapitel 4) getestet, wodurch sich die Evaluation in fünf Bereiche aufteilt. Die ersten drei Bereiche sind kennzeichnend durch Messungen zur Performanz der Datenübertragung (Abschnitt 7.1), des *Samplings* (Abschnitt 7.2) und der Visualisierung (Abschnitt 7.3). Darin werden auch weitere Messbedingungen der Evaluation aufgestellt, welche in der Anforderungsanalyse soweit noch nicht festgelegt waren. Weiterhin beinhaltet jeder Abschnitt eine Auswertung zu den Evaluationen. Der vierte Bereich (Abschnitt 7.4) evaluierter weitere funktionale und nicht-funktionale Anforderungen des Systems. Abschließend werden die gestellten Anforderungen durch die vorangegangen Evaluationen in einem Fazit (Abschnitt 7.5) auf ihre erfolgreiche Umsetzung geprüft.

Die Messungen fanden auf einem Acer Aspire TimelineX 4820TG statt. Der Rechner hat einen Intel Core i5-460M Prozessor mit 2,53 GHz und einem Hauptspeicher mit 4 GB DDR3-RAM. Das Backend und Frontend der PIPELINR Anwendung liefen auf dem selben Rechner, wodurch die Zeit der Netzwerkübertragung zu vernachlässigen ist.

### 7.1. Evaluation zur Datenübertragung

Die Anforderungen an das System legten Werte für die effiziente Datenübertragung (Tabelle 1) fest. Eine Datenübertragung beinhaltet die Zeit des Absendens einer Abfrage bis zur Ankunft der Antwort. Eine statistische Messung verschiedener Werte, aber vor allem der Durchschnittszeit zur Datenübertragung, soll das NODE.js Client Modul BENCH-REST bieten. Das Modul ermöglicht es, REST APIs einem Benchmark zu unterziehen. Dafür können Abfragen an den Service definiert werden, um diese abschließend mehrfach und über mehrere konkurrierende Verbindungen an den Service abzusenden. Die Messung liefert Messwerte zur Mindest-, Durchschnitts- und Maximalzeit der Datenübertragung. Ein weiterer Messwert, der P95-Wert, gibt an, unter welcher Grenze des Wertebereichs 95% aller Messwerte fallen. Da es oft zu Ausreißern von Maximalwerten in der Messung kam, wurde der P95-Wert anstelle des Maximalwerts bevorzugt. Weiterhin können in einer Messung die durchschnittlichen getätigten Abfragen pro Sekunde aufgenommen werden.

Im Folgenden werden die einzelnen Evaluationen zur Datenübertragung beschrieben und ausgewertet.

**Evaluation: Steigende Last** Die erste Evaluation der Datenübertragung erfasste Messungen für das Schreiben eines neuen Wertes in einer Pipeline bei einer kontinuierlichen

## 7.1. Evaluation zur Datenübertragung

---

Erhöhung der Abfragen (500 auf 10.000) an das PIPELINR Backend. Es war herauszufinden, ob das PIPELINR Backend ab einer bestimmten Anzahl von Abfragen, also Schreiboperationen, an Performanz verliert. In jeder Einzelmessung wurde bei einer gleichbleibenden Anzahl von vier konkurrierenden Verbindungen die Zahl der Abfragen erhöht.

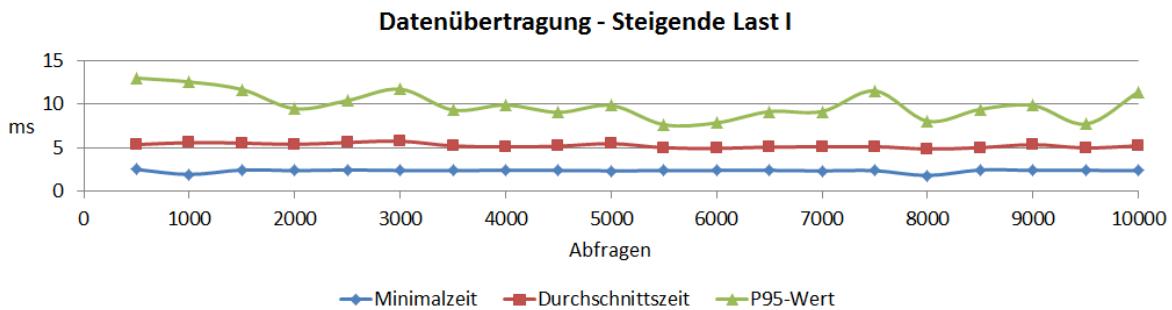


Abbildung 38: Evaluation: Steigende Last durch kontinuierliche Schreiboperationen neuer Werte in einer Pipeline I

Aus den Diagrammen (38, 39) werden die aufgenommenen Werte bei einer steigenden Last ersichtlich. Der Wertebereich in Diagramm 38 für die Zeiten der Datenübertragung ist ausgesprochen schmal und niedrig. Demnach unterliegen die Werte kaum einer Schwankung bei steigender Last. Die Werte der durchschnittlichen Datenübertragung sind mit 4,84 ms bis 5,73 ms merklich gering.

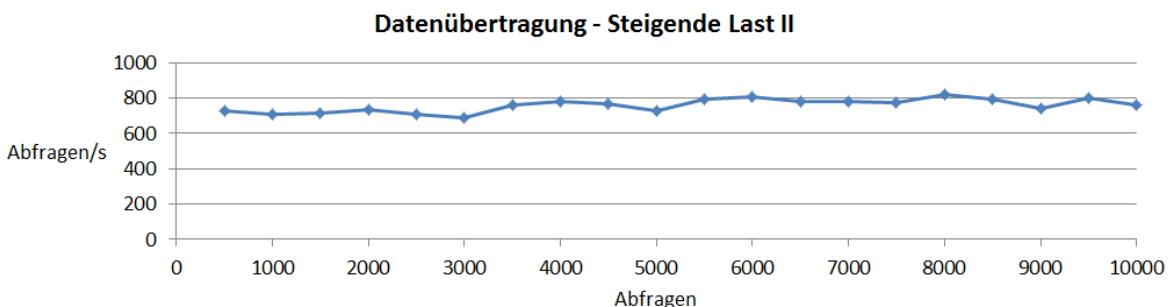


Abbildung 39: Evaluation: Steigende Last durch kontinuierliche Schreiboperationen neuer Werte in einer Pipeline II

Auch die getätigten Abfragen pro Sekunde bleiben zwischen den Werten 688 und 817 bei einer steigenden Last äußerst stabil (Diagramm 39).

Insgesamt zeigt diese Evaluation, dass das PIPELINR Backend bei dem Schreiben neuer Werte keinerlei Schwankungen bei einer steigenden Last bis zu 10.000 Werten unterliegt. 10.000 Werte können außerdem in einer Zeit von 12,903 Sekunden geschrieben werden.

**Evaluation: Abfrage großer Pipelines** Die zweite Evaluation zur Datenübertragung befasst sich mit der Abfrage einer in der Anzahl von Werten variierenden Pipeline. Die Zahl variiert zwischen 1000 und 15.000. Je individuelle Pipeline wurden jeweils 100 Abfragen durch eine konkurrierenden Verbindung gestellt, wodurch die Zeiten der Datenübertragung aus einer Messreihe ermittelt werden konnten. Die Werte sind aus dem Diagramm 40 zu entnehmen.

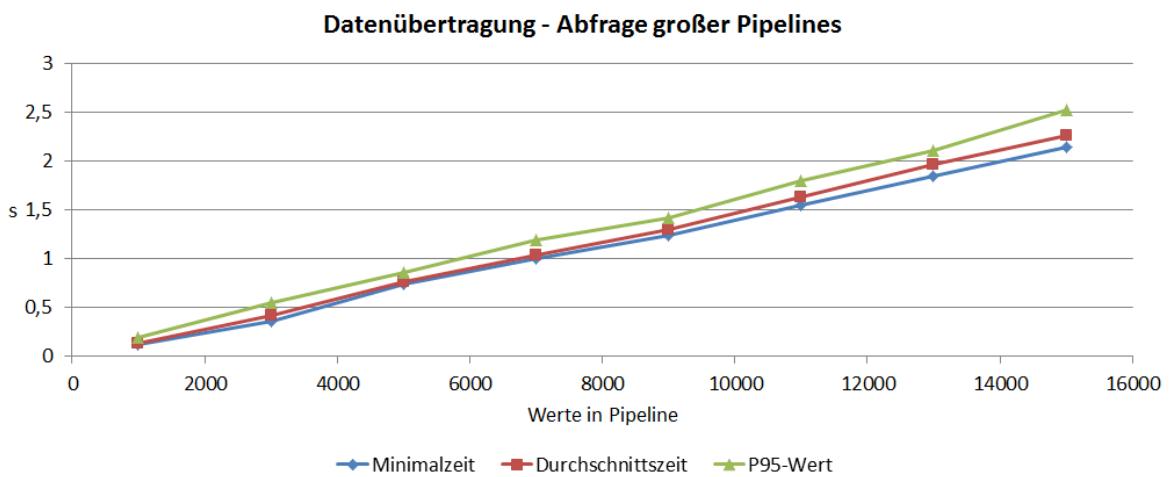


Abbildung 40: Evaluation - Abfrage großer Pipelines bei variierender Größe

Die Messreihen zeigen einen linearen Anstieg der Zeiten. Während für 1000 Werte in einer Pipeline die Durchschnittszeit einer Abfrage bei 135,44 ms liegt, beträgt die Zeit bei 15.000 Werten 2,261 s. Es ist mit einem Anstieg von ungefähr 150 ms bei einem Zuwachs von 1000 Werten zu rechnen.

Die nun merklich langsameren Abfragen von großen Datensätzen wurden zugunsten einer schnelleren Schreiboperation von Werten (Evaluation: Steigende Last) implementiert. Je nach Aufbau der Datenstruktur, ergaben sich unterschiedliche Kompromisse in den Lese- und Schreiboperationen.

**Auswertung** Die erste Evaluation lieferte für das Schreiben eines neuen Wertes in einen Datensatz durchschnittliche Zeiten in der Datenübertragung zwischen 4,84 ms bis 5,87

## 7.2. Evaluation zum Sampling

---

ms. In den Anforderungen wurde ein Wert von 40 ms angesetzt. Die erreichten Werte liegen somit deutlich unter der Zielsetzung. Die Möglichkeit einer solchen effizienten Schreiboperation ging auf Kosten einer langsameren Leseoperation für die Datensätze. Daher wurde in der zweiten Evaluation zur Abfrage einer großen Pipeline nur eine durchschnittliche Datenübertragung von 760,70 ms gegenüber einer aus den Anforderungen angestrebten Zeit von 500 ms bei 5000 Werten erreicht. Da die Abstimmung der Effizienz zwischen Lese- und Schreiboperation gezielt so gewählt wurde, sind die Ergebnisse der Evaluation zur Datenübertragung zufriedenstellend.

## 7.2. Evaluation zum Sampling

Die Evaluation zum *Sampling* wurde so bisher nicht in den Anforderungen bedacht. Dennoch ist es notwendig, die Performanz und Vorteile des *Samplings* für das System durch eine Evaluation aufzuzeigen. Das Ausdünnen von Datensätzen wurde bereits in der Implementierung erläutert. Ein sogenannter Job überprüft periodisch die Größe der Datensätze und dünnst diese gegebenenfalls durch *Sampling* aus. Für die Evaluation wurde das *Stratified sampling* mit einer Rate von 5 Sekunden verwendet. Im Folgenden werden die Evaluationen zum *Sampling* durchgeführt.

**Evaluation: Vergleich der Performanz** In dieser Evaluation galt es, die Performanz des Systems beim *Sampling* während einer starken Auslastung zu prüfen. Dafür wurden zwei Messungen mit jeweils 100.000 Abfragen durch vier konkurrierenden Verbindungen angestoßen. Eine Abfrage entspricht wie zuvor der Schreiboperation eines neuen Wertes in einem Datensatz. Während der Auslastung soll es für eine der zwei Versuche zum periodischen *Sampling* von 15 Sekunden kommen. Dadurch sollten Schwankungen in der Performanz durch die getätigten Abfragen pro Sekunde wahrzunehmen sein. Das Diagramm 41 zeigt beide Messungen mit den getätigten Abfragen pro Sekunde über eine Zeitspanne in Sekunden.

Durch das periodische *Sampling* zeigt das Diagramm 41 in gleichen Abständen von 15 Sekunden Ausreißer in der Performanz für die Messung mit *Sampling* auf. Die Ausreißer variieren von 160 Abfragen pro Sekunde bis zu einem kaum merklichen Unterschied zu der Messung ohne *Sampling*. Die 100.000 Abfragen wurde in einer Zeit von 204,36 Sekunden abgeschlossen. Auf der anderen Seite unterliegt die Messung ohne *Sampling* nur einer geringen Schwankung. Diese wurde in 135,45 Sekunden abgeschlossen.

Das *Sampling* selbst nimmt einiges an Rechenleistung in Anspruch, weswegen zu dieser Zeit weniger Abfragen pro Sekunde der eigentlichen Messung getätigten werden konnten. Aus der Evaluation wird ersichtlich, dass ein *Sampling* während des Schreibens von Werten die Performanz beeinflusst. Es sollte jedoch festgehalten werden, dass trotz

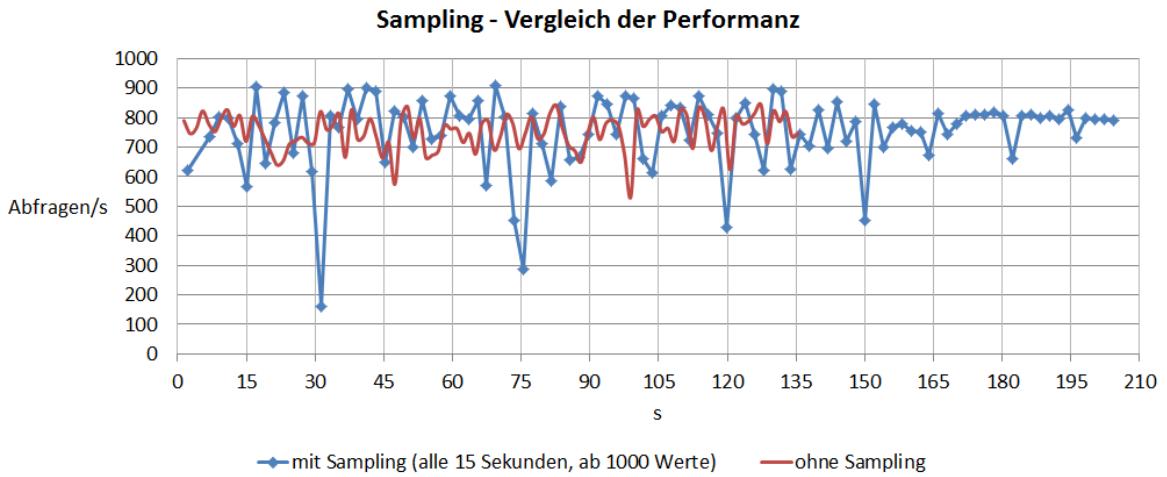


Abbildung 41: Evaluation - Vergleich der Performanz zweier Pipelines (mit und ohne Sampling) bei kontinuierlicher Last durch Schreiboperationen neuer Werte in einer Pipeline

*Samplings* im schlechtesten Fall immer noch 160 Werte pro Sekunde in einem Datensatz geschrieben werden konnten.

**Evaluation: Bearbeitungszeit** Die zweite Evaluation zum *Sampling* soll die Bearbeitungszeit einer Pipeline aufzeigen. Dafür wurden Pipelines, welche in der Größe variieren, mehrmals einem *Sampling* unterzogen, um die gemittelte Bearbeitungszeit zu errechnen. Die Werte in einer Pipeline verteilten sich bei allen Pipelines immer über die gleiche Zeitspanne. Je mehr Werte eine Pipeline beinhaltete, desto dichter wurden diese in der Zeitspanne angeordnet. Diagramm 42 zeigt die Bearbeitungszeiten für Pipelines mit 10.000 bis 100.000 Werten.

Aus dem Diagramm wird ein linearer Anstieg der Bearbeitungszeit bei größer werdenden Pipelines deutlich. Während das *Sampling* für 10.000 Werte eine Zeit von 3,740 s benötigte, waren es für 100.000 Werte 35,875 s. Die Bearbeitungszeit stieg um 3,6 s bei einem Zuwachs von 10.000 Werten. Trotz linearen Anstiegs, sollte möglichst oft ein *Sampling* durchgeführt werden, um die Bearbeitungszeiten kurz zu halten.

**Evaluation: Speicherauslastung** Die letzte Evaluation zum *Sampling* zeigt die Anzahl der maximalen Werte in einer Pipeline bei einem periodischen *Sampling* von 15 Sekunden. Hier wurde die Anwendung durch 50.000 Schreiboperationen bei vier konkurrieren-

## 7.2. Evaluation zum Sampling

---

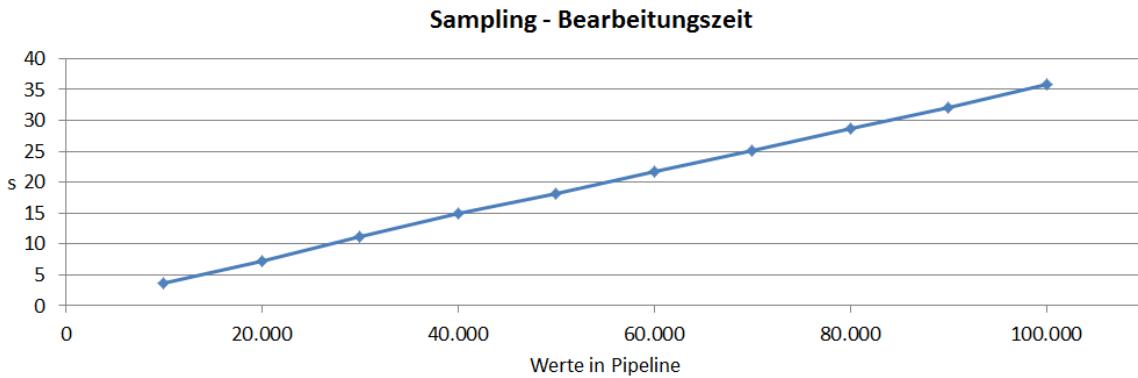


Abbildung 42: Evaluation - Bearbeitungszeit des Samplings bei variierender Anzahl von Werten in der Pipeline

den Verbindungen ausgelastet. Dadurch kamen über den Messversuch durchschnittlich 507,56 Abfragen pro Sekunde zustande, wobei eine durchschnittliche Abfrage 7,83 ms benötigte. Die 50.000 Abfragen wurden nach 99,149 s getätigten. In dieser Zeit kam es an 6 Stellen (Diagramm 43) zum *Sampling*. Beim Auslösen der Datenreduzierung wurde die aktuelle Anzahl an Werten ermittelt. Demnach kamen die Werte nie über einen Bereich von 8000 bei einem periodischen *Sampling* von 15 Sekunden. Zwischen den *Samplings* wurden weitere Messwerte für die Anzahl an Werten der Pipeline ermittelt, um im Diagramm einen ungefähren Verlauf der Wertanzahl zu vermitteln.

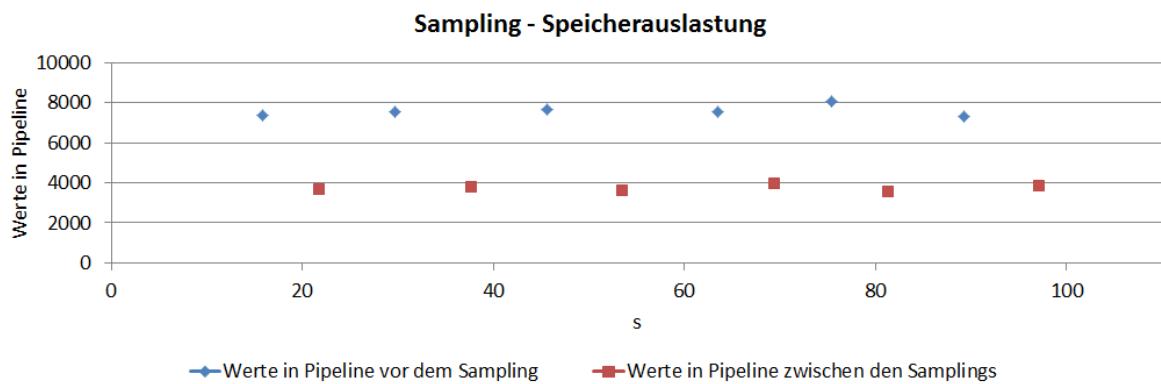


Abbildung 43: Evaluation - Maximale Speicherauslastungen (Anzahl der Werte) einer Pipeline bei periodischem Sampling

**Auswertung** Die Evaluation zum *Sampling* wurde so bisher nicht in den Anforderungen bedacht. Nach der Implementierung wurde jedoch deutlich, dass auch das *Sampling* evaluiert werden sollte. Die erste Evaluation zeigt, dass auch unter einer hohen Last das *Sampling* nur für kurze Zeit die Performanz des Systems beeinflusst. Auch waren die Beeinflussungen teilweise sehr gering. Wie die zweite Evaluation zeigt, konnte weiterhin die Bearbeitungszeit des *Samplings* bei einer dauerhaften Verwendung gering gehalten werden. Daneben werden weitere Vorteile durch das *Sampling* in der PIPELINR Anwendung erzielt: Die dritte Evaluation zeigt, dass durch ein periodisches Ausdünnen der Daten die Speicherauslastung des Systems gering gehalten werden kann. Dadurch wird auch, wie aus der ersten Evaluationen zur Datenübertragung (Abschnitt 7.1) ersichtlich, dass die Abfrage kleinerer Pipelines deutlich effizienter vollzogen werden kann. Weiterhin besteht der Verdacht, die Darstellungszeit einer Pipeline oder eines neuen Wertes durch ein regelmäßiges *Sampling* zu vermindern. Die folgende Evaluation zur Visualisierung soll unter anderem diesen Verdacht untersuchen.

### 7.3. Evaluation zur Visualisierung

Die Evaluation zur Visualisierung fand im Browser Chrome von Google in der Version 37 statt, da in diesem Browser auch das PIPELINR Frontend entwickelt wurde. Nach den Anforderungen sollten die zwei folgenden Evaluationen, Darstellen einer Pipeline und Darstellen eines neuen Wertes, durchgeführt werden, welche im Folgenden beschrieben und ausgewertet werden.

**Evaluation: Darstellen einer Pipeline** Die erste Evaluation zur Datenvisualisierung befasst sich mit der Performanz zur Darstellung einer Pipeline mit 1000 Werten. Zwar wird in den Anforderungen von nur einem Datensatz ausgegangen, da es in der Implementierung aber zwei verschiedene Visualisierungsformen für Datensätze gibt, soll nun eine Pipeline mit zwei Datensätzen und jeweils 500 Werten dargestellt werden. Darüber hinaus soll die Anzahl der dargestellten Werte erhöht werden, um die Performanz bei einem wachsenden Datensatz beurteilen zu können.

Gemessen wurde die Zeit zwischen Ankunft der Pipeline vom PIPELINR Backend und vollständiger Darstellung der Werte als Visualisierung. Durch den Einsatz der *Directives* von ANGULAR.JS, lassen sich Events für die ankommenden Daten und der abgeschlossenen Visualisierung finden. Dementsprechend wurde die Messreihe für eine steigende Anzahl von Werten in der Pipeline aufgestellt. Das Ergebnis ist dem Diagramm 44 zu entnehmen.

Das Diagramm zeigt eine linear wachsende Darstellungszeit bei größer werdenden Pipelines. Während bei 1000 Werten die Durchschnittszeit 0,564 s beträgt, wächst die Zeit

### 7.3. Evaluation zur Visualisierung

---

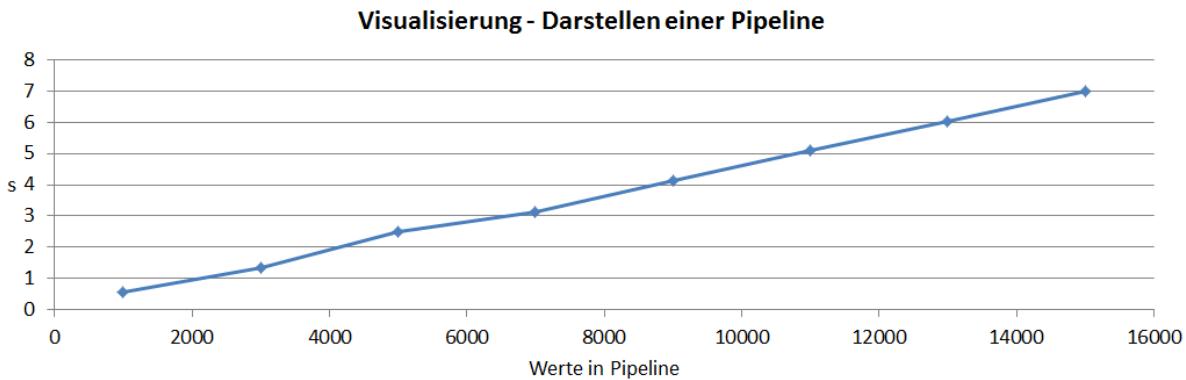


Abbildung 44: Evaluation - Darstellen einer Pipeline bei variierender Größe

linear auf 7,005 s bei 15.000 Werten an. Alle 1000 Werte wächst die Darstellungszeit um ungefähr 500 ms.

**Evaluation: Darstellen eines neuen Wertes** Angesichts der Echtzeitfähigkeit sollte weiterhin die benötigte Darstellungszeit eines neuen Wertes in einer visualisierten Pipeline evaluiert werden. Als Ausgangspunkt besteht hier die Pipeline aus zwei Datensätzen mit jeweils 500 Werten. Weiterhin sollen die Werte, wie in der vorangegangenen Evaluation, zur Bildung einer Messreihe erhöht werden, da davon auszugehen ist, dass die Darstellungszeit eines neuen Wertes mit der Anzahl der insgesamt darzustellenden Werte ansteigt.

Die Messung wurde für ein voll ausgelastetes PIPELINR Frontend erstellt. Demnach wurde das PIPELINR Backend mit neuen Werten regelrecht überflutet, um gleichzeitig das PIPELINR Frontend an die Grenzen der Auslastung durch die Benachrichtigungen der Websockets zu bringen. Demnach kam es durch die neuen Werte zu einem Stau an Aktualisierungen der Visualisierung im PIPELINR Frontend, wodurch erst nach dem Verarbeiten einer Aktualisierung die nächste Aktualisierung angestoßen werden konnte. Wie in der vorherigen Evaluation konnten auch hier Events in der *Directive* ausfindig gemacht werden, um die Zeit zwischen Ankunft des neuen Wertes und Fertigstellung der Visualisierung zu bemessen. Diagramm 45 zeigt diese Messreihe durch eine blaue Linie. Die Werte steigen linear von 58 ms bei 1000 Werten bis zu 682 ms bei 15.000 Werten an.

Weiterhin konnte zur gleichen Zeit eine Messreihe für die vollständigen Aktualisierungen aufgenommen werden, da das Frontend vollständig ausgelastet war und somit die Berechnungszeit zwischen zwei eintreffenden Werten bemessen werden konnte. Das Er-

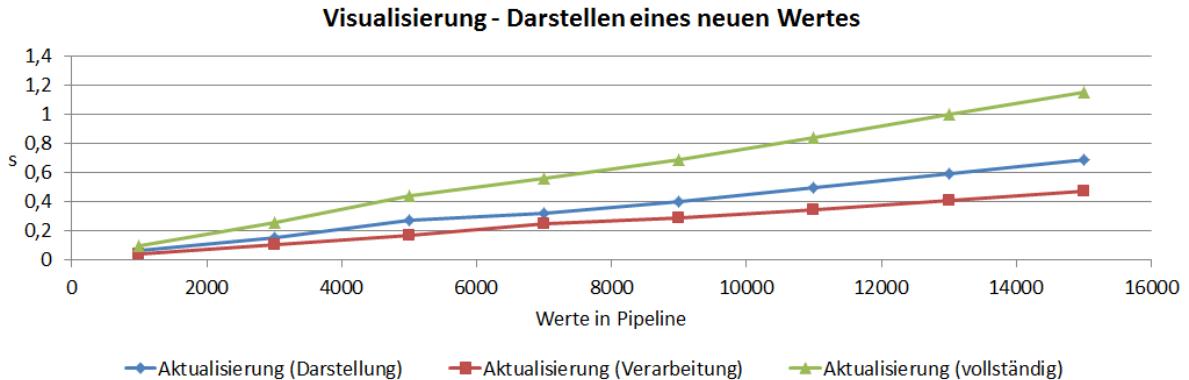


Abbildung 45: Evaluation - Darstellen eines neuen Wertes bei variierender Größe

gebnis ist im Diagramm 45 als grüne Linie zu sehen. Die Zeiten reichen von 97 ms bei 1000 vorhandenen Werten bis zu einer Dauer von 1,153 s bei 15.000 vorhandenen Werten.

Da die beiden vorangehenden Messreihen zeitgleich aufgenommen wurden, konnte aus der Differenz zwischen vollständiger Aktualisierungszeit und der Zeit einer aktualisierten Darstellung die verbleibende Verarbeitungszeit im PIPELINR Frontend ermittelt werden. Das Ergebnis ist auch im Diagramm 45 als rote Linie zu sehen.

**Auswertung** Die erste Evaluation, Darstellen einer Pipeline, erreichte Durchschnittszeiten zwischen 0,477 s bei 1000 Werten bis 7,005 s bei 15.000 Werten. Die zum Ziel gesetzte Zeit von 2000 ms bei 5000 Werten wurde im Vergleich zur gemessenen Zeit von 2,496 s nicht erreicht. Dennoch ist zu erkennen, dass bei größeren Datenaufkommen die Darstellungszeiten nur linear ansteigen.

In der zweiten Evaluation, Darstellen eines neuen Wertes, ist insgesamt zu erkennen, dass die Verarbeitungszeit neben der Darstellungszeit einen großen Anteil der gesamten Aktualisierung ausmacht. Weiterhin steigen alle Messreihen linear an. Es wurden durchschnittliche Zeiten zur Aktualisierung von 97 ms bei 1000 Werten bis zu 1,153 s bei 15.000 Werten erreicht. Die zum Ziel gesetzte Zeit von 500 ms bei 5000 Werten wurden mit einem Wert von 435 ms erreicht.

Die Messwerte lieferten weiterhin wichtige Erkenntnisse darüber, wie das Aktualisierungsverhalten des PIPELINR Frontends verbessert werden könnte. Im Falle einer Überflutung von Aktualisierungen, kommt es im Frontend zum Stau. Demnach kann beispielsweise, wie in der Evaluation, nur alle 0,435 ms bei 5000 Werten eine Aktualisierung vorgenommen werden. Eine angepasste Implementierung würde vorsehen, dieses Verhal-

#### 7.4. Evaluation der weiteren Anforderungen

---

ten auszunutzen, um die Aktualisierungen zwischenzuspeichern. Zwischengespeicherte Werte, die noch nicht visualisiert wurden, könnten dann bei 5000 Werten alle 0,435 ms dargestellt werden. Das Verhalten würde sich dynamisch an größer werdende Pipelines anpassen lassen.

Auch das Verhalten des *Samplings* könnte durch das Wissen der Evaluation zur Visualisierung dynamisch angepasst werden. Demnach könnten durch Nutzerstudien akzeptable Zeiten für das Darstellen eines neuen Wertes und das Darstellen einer Pipeline ermittelt werden, um das *Sampling* aus dem Modul zur Datenreduzierung ab einer entsprechenden Größe der Pipeline anzustoßen. Dadurch würden nur Pipelines mit einer akzeptablen Darstellungs- und Aktualisierungszeit vom PIPELINR Backend abgeholt werden. Demnach wäre es nicht länger notwendig, zu großen Datensätzen einer Pipeline zu Lasten der Nutzerakzeptanz darzustellen.

Beide Evaluationen zur Datenvisualisierung geben essentielle Erkenntnisse über mögliche Verbesserungen der PIPELINR Anwendung. Die Ergebnisse können zu aufgezeigten Erweiterungen im System beitragen.

## 7.4. Evaluation der weiteren Anforderungen

In diesen Abschnitt werden weitere funktionale und nicht-funktionale Anforderungen aus dem Kapitel zur Anforderungsanalyse (Kapitel 4) auf ihre Implementierung evaluiert.

**Unabhängigkeit** Die PIPELINR Anwendung wurde als eigenständiges System konzipiert. Schnittstellen liefern Zugang für Service-Konsumenten, um neue Pipelines sowie Datensätze anzulegen und diese mit Werten zu füllen. Auch Manipulationen der Daten sind möglich. Weiterhin zeigt die bisherige Evaluation, dass auch mehrere konkurrierende Service-Konsumenten die PIPELINR Anwendung zeitgleich nutzen können. [NF0010]

**Performanz** Die Datenstruktur wurde wie im Konzept umgesetzt (Abbildung 28). Vor allem die Datenstrukturen Pipeline, Datensatz und Wert wurden lose durch Referenzierungen gekoppelt. Zuerst wurde in der Implementierung eine bidirektionale Referenzierung zwischen Datensatz und Wert vorgesehen. Da es dadurch aber zu einem stetig wachsenden Array aus Referenzierungen im Datensatz kam, litt die Performanz für Schreiboperationen bei neu hinzukommenden Werten stark darunter. Deshalb wurde von der bidirektionalen auf eine unidirektionale Referenzierung gewechselt. Nur die Werte haben Kenntnisse über ihren zugehörigen Datensatz. Dadurch kommen schlussendlich auch die Messungen zur Evaluation der Datenübertragung zustande. Schreiboperationen sind nun sehr schnell, während Leseoperationen einer Pipeline mit Werten deutlich

langsamer sind. [NF0020]

Weiterhin wurde stets bedacht, nur die nötigsten Daten zu übertragen, um dadurch den Kommunikationskanal so schmal wie möglich zu halten. Beispielsweise werden zur Übersichtsdarstellung aller Pipelines keine Werte mitgesendet. [NF0030]

**Modularität und Module** Die Modularität wurde vor allem durch ein erweiterbares Modul zur Datenreduzierung geschaffen. Dieses Modul hält bereits verschiedene Funktionen bereit. Die geforderte permanente Datenreduzierung wird durch einen periodisch angestoßenen Job zum *Sampling* verrichtet. [0F0010] Weiterhin ist es möglich, die Pipeline auf eine Zeitspanne zu begrenzen. Dafür wurde das *Dynamic Query Filter* implementiert. [0F0030] Abschließend wurde das *Sampling* implementiert, um Daten nach bestimmten Verfahren auszündnen zu können. [0F0020]

Wie im Konzept beschrieben, entfällt das Modul zur Datenanalyse. Dafür wurden *Directives* im Dashboard mit Analysefunktionen erweitert. Demnach ist es möglich, über eine Trendlinie einen möglichen zukünftigen Verlauf auszumachen [1F0050] und weitere statistische Daten wie Standardabweichung, Varianz und Durchschnittswert auszumachen. [0F0060]

Andere Stellen in der Anwendung weisen ebenso eine hohe Modularität auf. Im PIPELINR Frontend wird ein Dashboard für die Hauptvisualisierungen der Datensätze verwendet. Da es auf dem Prinzip der zusammengesetzten *Directives* aufbaut, können weitere *Directives* implementiert und dem Dashboard hinzugefügt werden. Auch die Abfrage einer Pipeline kann, wie beschrieben, durch optionale Parameter zur Datenreduzierung angereichert werden. Da die Module im PIPELINR Backend erweiterbar sind und die Auflösung der optionalen Parameter generisch erfolgt, können unkompliziert weitere optionale Parameter zur Abfrage hinzugefügt werden. [NF0040]

**Visualisierung** Die Hauptvisualisierungen wurden durch *Small Multiples* in Form von *Line Graphs* für die metrischen Datensätze und einem *Point Graph* für einen nominalen Datensatz auf zeitbasierte Visualisierungsformen mit Wiedererkennungswert abgebildet. Für diese Visualisierungen ist der Kontextverlust bei neu hinzukommenden Daten sehr gering (Abschnitt 3.3). Auch wurde auf die Einfärbungen und Formen geachtet, um visuelle Kodierungen zu schaffen. [0F0070]

Weiterhin wurde mit der Zooming-Leiste eine Visualisierung zur Unterstützung von Interaktionen geschaffen. Diese bietet außerdem einen Überblick über den gesamten Zeitraum in Form aller nominalen Daten. [1F0080]

## 7.5. Fazit

---

**Interaktionstechniken** In den Anforderungen wurden auch Interaktionstechniken zur Exploration der Daten gesucht. Demnach ist es nun möglich, im Verarbeitungsfenster eine Zeitspanne für die abgefragte Pipeline festzulegen. Dadurch wird die Pipeline auf ein Intervall reduziert. [0F0090] Weiterhin ermöglicht eine Legende das interaktive Filtern von nominalen Daten [0F0100] und eine Selektion der Datensätze die Bereitstellung nur ausgewählter Zeitreihen [0F0110]. Durch die Zooming-Leiste wird der Nutzer zu einem *Zooming* und *Panning* befähigt. [0F0120] Abschließend wurden nominale Werte aus unterschiedlichen Visualisierungen durch ein *Linking* miteinander verknüpft, um die Nutzer bei der Zuordnung von Datenwertpaaren zu unterstützen. [1F0140]

**Schnittstelle** Alle geforderten Schnittstellen aus den Anforderungen konnten implementiert werden. [0F0170, 0F0180, 0F0190, 0F0200] Auch das Abholen und die Erstellung einer Pipeline mit zusätzlichen Parametern ist möglich. [0F0210, 0F0215]

## 7.5. Fazit

Abschließend zum Kapitel der Evaluation zeigt Tabelle 4 zusammenfassend alle erreichten, teilweise erfüllten und unerreichten Anforderungen in grüner, gelber und roter Markierung auf. Die Anforderungen [0F0230], [0F0240] erreichten nicht die geforderten Übertragungszeiten, überschreiten diese jedoch nur gering und könnten durch Erweiterungen der Anwendung verbessert werden. Weitergehend wurden einige Kann-Kriterien aus den Anforderungen nicht erfüllt ([1F0015], [1F0040], [1F0050], [1F0125]). Diese umfassen vor allem fortgeschrittene Analysemethoden und dynamisches *Sampling* in der Visualisierung durch Interaktionsmöglichkeiten. Daneben wurden die statistischen Analysen durch das Modul zur Datenanalyse [0F0060] als teilweise erfüllt markiert, da diese Implementierung entgegen der Anforderungen ins PIPELINR Frontend verlegt und als Erweiterung in den *Directives* implementiert wurden. Auch wurde so das *Forecasting* nur als Trendlinie umgesetzt und nicht durch zukünftige Werte. Zusammenfassend konnten alle Soll-Kriterien zufriedenstellend erfüllt werden.

Nr.	Kategorie	Beschreibung
[NF0010]	Unabhängigkeit	eigenständiges System
[NF0020]	Performanz	effektive Datenstruktur
[NF0030]	Performanz	schmaler Kommunikationskanal
[NF0040]	Modularität	austauschbare, erweiterbare Module
[0F0010]	Modul zur Datenreduzierung	permanente Reduzierung
[1F0015]	Modul zur Datenreduzierung	dynamische Reduzierung
[0F0020]	Modul zur Datenreduzierung	Sampling
[0F0030]	Modul zur Datenreduzierung	Auswahl der Zeitspanne
[1F0040]	Modul zur Datenanalyse	Outlier Analysis
[1F0050]	Modul zur Datenanalyse	Forecasting
[0F0060]	Modul zur Datenanalyse	statistische Analyse
[0F0070]	Visualisierung	grundlegende Kriterien
[1F0080]	Visualisierung	erweiterte Visualisierungen
[0F0090]	Interaktionstechniken	interaktive Datenreduzierung
[0F0100]	Interaktionstechniken	Filterung
[0F0110]	Interaktionstechniken	Selektion der Datensätze
[0F0120]	Interaktionstechniken	Zooming und Panning
[1F0125]	Interaktionstechniken	dynamisches Sampling
[1F0140]	Interaktionstechniken	Linking und Brushing
[0F0170]	Schnittstellen	Erstellung eines Datendepots
[0F0180]	Schnittstellen	Füllen des Datendepots mit Daten
[0F0190]	Schnittstellen	Löschen eines Datendepots
[0F0200]	Schnittstellen	Abfrage d. Daten eines Datendepots
[0F0210]	Schnittstellen	optionale Parameter bei Abfrage
[0F0215]	Schnittstellen	optionale Parameter bei Erstellung
[0F0220]	Datenübertragung	Schreiben eines neuen Wertes
[0F0230]	Datenübertragung	Abfrage einer Pipeline
[0F0240]	Datenvisualisierung	Darstellen einer Pipeline
[0F0250]	Datenvisualisierung	Darstellen eines neuen Wertes

Tabelle 4: Übersicht der erfüllten Anforderungen



## 8. Ausblick

An mehreren Stellen dieser wissenschaftlichen Arbeit gab es Entwürfe, welche schlussendlich nicht umgesetzt werden konnten. Auch existieren nach der Implementierung von PIPELINR weitere Vorstellungen zur Erweiterung der Anwendung, vorwiegend in den behandelten Bereichen der Datenreduzierung, Datenanalyse, Visualisierung und Interaktion.

Ein dynamisches *Sampling* wurde schon in der Anforderungsanalyse bedacht. Demnach würde bei der initialen Abfrage einer zu großen Pipeline nur eine ausgedünnte Variante im PIPELINR Frontend dargestellt werden. Erst durch ein *Zooming* würden Bereiche der Pipeline detaillierter nachgeladen werden, gleichwohl würde sich die Anzahl der dargestellten Werte nur gering verändern, da die visualisierten Werte am Rand durch das Hineinzoomen verschwinden würden. Der Vorteil wäre die schnellere Darstellung der Visualisierung und agilere Interaktion durch weniger dargestellte Werte. Nutzerstudien könnten eine optimale Größe für die Wahrnehmung von weichen Interaktionen ausmachen, um nur so viele Datenpunkte zu visualisieren, damit das Nutzerempfinden am angenehmsten ist.

Weiterhin wurden im Konzept generisch viele Datensätze, ungeachtet der metrischen oder nominalen Natur, in einer Pipeline angedacht. Jedoch wurde die Visualisierung mit nur einem nominalen Datensatz umgesetzt. Eine generische Anzahl ist dennoch durch eine Erweiterung in der Visualisierung möglich. Genauso wie für die metrischen Datensätze unabhängig viele *Line Graphs* als *Small Multiples* erzeugt werden können, gilt das Gleiche auch für die nominalen Datensätze. Darüber hinaus werden die nominalen Werte gegenüber den metrischen Daten auch im Overview-Bereich (Zooming-Leiste) zusammengefasst dargestellt. Da diese Aneinanderreihung der Werte nur eindimensional stattfindet, könnten weitere nominale Datensätze in eine zweite Dimension übereinander gestapelt (*Stacking*) werden. Abschließend müsste die bereits existierende Tabelle aus nominalen Werten lediglich alle Datensätze vereinen und anzeigen.

Die Datenanalyse wurde bisher in den *Line Graphs* der metrischen Datensätze in Form von statistischen Größen visualisiert. Ferner existieren weitere Analysen aus den Grundlagen wie das *Forecasting* und die *Outlier Analyse*, welche in den *Line Graphs* integrierbar wären. Daneben können Analysen auch eigenständige Visualisierungen formen. Wie bereits im Konzept angedacht, würden diese Analysevisualisierungen horizontal neben den *Line Graphs* angeordnet werden. Um dem Nutzer eine Auswahl an Analysen zu bieten, könnte ein Plugin-System das interaktive Hinzufügen und Entfernen von Visualisierungen zur Analyse ermöglichen. Diese Erweiterung bedarf eines passenden Plugin-Systems sowie weiterer Analysearten und -visualisierungen.

Neben der Datenanalyse könnte auch die Datenreduzierung erweitert werden. Für das

---

*Sampling* existieren weitere Algorithmen, welche durch das Modul zur Datenreduzierung flexibel ergänzt werden könnten.

Weiterhin wurden in den Grundlagen verschiedene Interaktionsmöglichkeiten der *Visual Analytics* Anwendungen vorgestellt. Eine vielversprechende Interaktion für PIPELINR wäre das Annotieren in den visualisierten Datensätzen. Danach könnten Anwender an relevanten Stellen Anmerkungen hinterlassen, welche zur späteren Analyseaufnahme unterstützen oder die Kommunikation zwischen Nutzern erleichtern. Weitere nützliche Interaktionen aus den Grundlagen wären das *Share* und *Record* von Analyseschritten.

Grundsätzlich wurde die Analyse von einzelnen Datensätzen einer Pipeline ermöglicht. Nur durch einen Vergleich der visualisierten Graphen ist eine Analyse mehrere Datensätze realisierbar. Weiterführende Algorithmen, wie die Diskrete Fourier-Transformation, könnten zwei metrische Datensätze angeleichen, um den Vergleich zu vereinfachen. Auch wären so automatische Vergleiche möglich. Daran angeknüpft könnten vollständige Pipelines mit ihren Datensätzen untereinander verglichen werden. Einige wissenschaftliche Arbeiten [Wu et al., 2000] beschäftigen sich mit dieser Art der übergreifenden automatischen Analyse zum Ausfindig machen von Zusammenhängen.

Abschließend wurden in der Evaluation schon einige Erweiterungen der Anwendung bedacht. Da die Visualisierung nur eine bestimmte Anzahl von neuen Werten pro Sekunde darstellen kann, könnte ein *Caching* im Frontend Staus von Aktualisierungen auflösen. Demnach würden alle Werte, welche in einem Intervall aktualisiert werden sollen, zwischengespeichert und in einer Aktualisierung hinzugefügt werden. Weiterhin beansprucht die Darstellung großer Pipelines eine zu lange Zeit. Danach könnte ein dynamisches *Sampling* die Pipelines für eine optimale Darstellungszeit ausdünnen.

Zusammenfassend lässt sich festhalten, dass es viele Erweiterungen in verschiedenen Ebenen für die PIPELINR Anwendung gibt. Demnach müssten auch die Forschungsgebiete für übergreifende Datenanalyse, weitere Methoden der Datenanalyse, Datenreduzierung, Nutzerermpfinden bei Visualisierungen und Interaktionen tiefgründiger eruiert werden.

## 9. Zusammenfassung

Diese Arbeit legt den Fokus auf die Umsetzung eines Systems zur Echtzeitvisualisierung großer Datenmengen. Während der Titel dieser Masterarbeit kurz und verständlich das Thema auf ein Minimum reduziert, wird das volle Spektrum erst in den einzelnen Kapiteln der Arbeit ersichtlich. Verständlicher wird dies auch beim Auflösen des Titels *Echtzeitvisualisierung für große Datenmengen*. Das Wort *Echtzeitvisualisierung* lässt sich in die zwei Themengebiete Darstellung und Kommunikation unterteilen. Daneben gibt die Wortgruppe *für große Datenmengen* Hinweise auf die zwei weiteren Themengebiete Datenverarbeitung und -speicherung. Im wesentlichen werden durch diese Begrifflichkeiten, von der Darstellung über die Kommunikation und Verarbeitung bis zur Speicherung, eine Vielzahl von Ebenen einer modernen Anwendung behandelt, welche es in dieser wissenschaftlichen Arbeit zu untersuchen, konzipieren und implementieren gilt.

Die Grundlagen (Kapitel 2) beschäftigen sich hauptsächlich mit den Themen Datenverarbeitung und Datenvisualisierung.

Im Bereich der Datenverarbeitung gibt es eine Einführung zum *Data Mining* sowie zu Datenbeständen bestehend aus *Data Warehouses*, relationalen Datenbanken und weiteren Datentypen. Zwei weitere Abschnitte gehen auf die Themen Datenanalyse und Datenreduktion ein, in welchen Methoden zur Analyse, wie das *Forecasting* und die *Outlier Analysis*, zum Anreichern von Daten und Methoden zur Reduzierung von Daten, wie das *Dynamic Query Filter* und *Sampling*, behandelt werden.

Neben der Datenverarbeitung werden im Bereich der Datenvisualisierung verschiedene Datenstrukturen sowie die darauf anzuwendenden Visualisierungsformen hauptsächlich für zeitbasierte Daten näher gebracht. Primär wird das Kriterium der Platzersparnis und Vergleichbarkeit bei Visualisierungen tiefgründig behandelt. Im Anschluss finden sich eine Reihe von Interaktionstechniken zur Exploration von Daten.

Die verwandten Arbeiten (Kapitel 3) geben Aufschluss über die Verarbeitung von Daten in verteilten Anwendungen, weitere Visualisierungsformen für multivariate Daten, Kontextveränderungen bei Echtzeitvisualisierungen und gesamte Systeme zur Datenexploration von zeitbasierten Daten.

Der erste Abschnitt zur Verarbeitung von Daten in verteilten Anwendungen behandelt ein Spektrum von Programmiermodellen wie *MapReduce* bis hin zu verteilten Systemen für große Datenmengen wie *HADOOP*. In diesem Bereich konnten viele Erkenntnisse zur effizienten Datenverarbeitung durch *Caching*, *batch processing*, *realtime processing* und *ad-hoc Analysen* gesammelt werden.

Der Abschnitt zur Visualisierung von multivariaten Daten behandelt weitere fortgeschrittenen Visualisierungsformen. Daneben zeigt der Abschnitt zu den Kontextveränderungen bei Echtzeitvisualisierungen essentielle Merkmale verschiedener Visualisierungsformen

---

beim Hinzukommen neuer Daten auf.

Der letzte Abschnitt in den verwandten Arbeiten zeigt relevante Systeme zur Echtzeitvisualisierung von großen Datenmengen. In diesem Bereich konnten Einblicke zur Konzeption eines Dashboards aus Visualisierungen, Interaktionsmöglichkeiten und Analyseverfahren gesammelt werden.

Insgesamt lieferten die Grundlagen (Kapitel 2) und verwandten Arbeiten (Kapitel 3) wichtige Erkenntnisse für das spätere Konzept und die Implementierung der eigenen Anwendung.

In den Grundlagen (Kapitel 2) wurde auch das NESSEE Projekt als Anwendungsfall für diese Arbeit vorgestellt. NESSEE ist eine Anwendung für die Emulation von Netzwerk- und Anwendungsverhalten für verteilte Systeme. Als Anwendungsfall können Echtzeitdaten gesammelt und zum in dieser Arbeit gesuchten System zur Echtzeitvisualisierung von großen Datenmengen übertragen, sowie dort verarbeitet, analysiert und visualisiert werden.

In der weiteren Anforderungsanalyse (Kapitel 4) wird nochmals auf das NESSEE Projekt eingegangen. Demnach wird ein Anwendungsfall durchlaufen, um Anforderungen an das gesuchte System zu extrahieren. Weiterhin werden nicht-funktionale und funktionale Anforderungen an das System zur Echtzeitvisualisierung von großen Datenmengen festgelegt.

Das Konzept (Kapitel 5) bezieht gewonnene Erkenntnisse aus den Grundlagen und verwandten Arbeiten, um die in der Anforderungsanalyse festgelegten Kriterien umzusetzen. Der Entwurf zur Architektur gibt einen ersten konzeptionellen Überblick der PIPERLINR genannten Anwendung, der Beziehung zu Service-Konsumenten, Schnittstellen und Anwendungsbereiche. Die Bereiche teilen sich in das Backend und Frontend der Anwendung auf.

Im Backend sollten zuerst die lose strukturierten Datenstrukturen einer Pipeline festgelegt werden. Danach wurden die Schnittstellen zum Backend für das PIPERLINR Frontend und Service-Konsumenten konzipiert. Das Modul zur Datenreduzierung wurde hauptsächlich durch die Grundlagen dieser Arbeit inspiriert. Im letzten Teil zum Backend wurde die Echtzeitübertragung durch ein *Publish-Subscribe* konzipiert, welches vom PIPERLINR Frontend zur Echtzeitvisualisierung der Daten genutzt werden soll.

Das Frontend teilt sich in die Bereiche Visualisierung und Interaktionsmöglichkeiten. Im Bereich zur Visualisierung wurde zuerst die Ausrichtung der Hauptvisualisierung evaluiert. Weiterhin wurde sich auf *Small Multiples* in Form von *Line Graphs* zur Visualisierung der Datensätze einer Pipeline geeinigt. Abschließend konnten weitere Visualisierungsformen für Analyseergebnisse der Daten bedacht werden. Die Interaktionsmöglichkeiten sollten zu aller erst Methoden aus dem Modul zur Datenreduzierung nutzen. Daher wurde eine Legende zur Filterung und weitere Elemente zum *Sampling* konzi-

piert. Weiterhin wurde ein *Zooming* und *Panning* zur Exploration großer Datenmengen und ein *Linking* und *Brushing* zur verbesserten Übersicht des Nutzers über die Daten entworfen.

PIPELINR basiert auf den Web-Technologien HTML, CSS und JavaScript. In der Implementierung (Kapitel 6) konnte vieles aus dem Konzept für die Anwendung umgesetzt werden.

Im Backend von PIPELINR wird NODE.JS für eine echtzeitfähige, skalierbare und performante Umsetzung einer Anwendung genutzt. Die Schnittstellen zum Service-Konsumenten und PIPELINR Frontend wurde mit dem REST Paradigma unter Verwendung des EXPRESS Frameworks implementiert. Weiterhin konnte eine hohe Modularität durch das NODE.JS Modulsystem geschaffen werden. Dies beinhaltet auch das Modul zur Datenreduzierung, welches erweiterbar ist und generisch Abfragen aus dem PIPELINR Frontend auflöst, während es Methoden zur Datenreduzierung wie *Sampling* und *Dynamic Query Filter* anbietet. Zur Speicherung wird MONGODB mit dem ODM Framework MONGOOSE verwendet. Abschließend macht es das SOCKET.IO Framework möglich, PIPELINR Backend und Frontend durch Websockets miteinander kommunizieren zu lassen.

Das Frontend von PIPELINR nutzt die REST Schnittstellen und Websockets zur Datenbeschaffung. Als Framework wurde ANGULAR.JS für eine klare Strukturierung der Anwendung gewählt. Die Visualisierungen wurden durch das Framework D3.JS als wiederverwendbare ANGULAR.JS Elemente implementiert. Das daraus resultierende Dashboard zur Echtzeitvisualisierung besteht aus einer Reihe dieser Elemente.

Ferner wurden die gesetzten Anforderungen evaluiert (Kapitel 7). Hier konnten erreichte und unerreichte Ziele ausfindig gemacht und durch Messreihen sowie Begründungen belegt werden. Das Resultat ist positiv für die entwickelte Anwendung ausgefallen.

Abschließend wurden im Ausblick (Kapitel 8) weitere mögliche Erweiterungen und die dafür zuständigen Forschungsgebiete zusammengefasst.

Insgesamt wurde eine Ausarbeitung und Anwendung geschaffen, welche viele wissenschaftliche Themen als eine Komposition vereint. Der Umgang mit großen Datenmengen und ihrer Möglichkeit der Echtzeitvisualisierung erfordert Kenntnisse in vielen Bereichen. Das Zusammenspiel zwischen Kommunikation und Visualisierung, Interaktion und Nutzerempfinden sowie Datenreduzierung, Analyse und Performanz muss effektiv ausbalanciert sein. Zudem ist ein großes Potenzial durch die vereinzelten Forschungsgebiete und einher gehende Erweiterungen in der zukünftigen Entwicklung von PIPELINR zu erkennen.

PIPELINR bietet Nutzern die Möglichkeit über eine API zeitbasierte Daten zur Anwendung zu überführen, um diese in einer Web-Anwendung durch Visualisierung, Interaktion und Analyse zu explorieren. Darüber hinaus werden Methoden für die Echtzeitvisualisierung und Datenreduzierung von PIPELINR bereitgestellt.



## A. Anhang

### A.1. Schnittstellenbeschreibung

```
1 POST: .. / pipelines
2
3 var data = { name: name, sampling: { task: task , perm:
4   perm, rate: rate } };
5 name: String
6 sampling: Object , null
7 task: Enumeration [ "frequencySampling" , "randomSampling" ,
8   "intervalSampling" ]
9 perm: Boolean
9 rate: Integer [ 1..99 ]
```

Listing 1: Erstellung einer Pipeline

```
1 POST: .. / pipelines /:id / datasets
2
3 var data = { key: key , type: type };
4
5 key: String
6 type: Enumeration [ "string" , "int" ]
```

Listing 2: Erstellung eines Datensatzes

```
1 POST: .. / pipelines /:id / datasets /:id / values
2
3 var data = { value: value , level: level , timestamp:
4   timestamp };
5 value: String
6 level: Enumeration [ "error" , "warning" ] – für Datensätze
7 vom Typ "string" , sonst null
7 timestamp: String [ "DD MM YYYY, HH:mm:ss :SSS" ]
```

Listing 3: Erstellung eines Wertes



## Literatur

- C. Ahlberg, C. Williamson, and B. Shneiderman. Dynamic queries for information exploration: An implementation and evaluation. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '92, pages 619–626, New York, NY, USA, 1992. ACM. ISBN 0-89791-513-5. doi: 10.1145/142750.143054. URL <http://doi.acm.org/10.1145/142750.143054>.
- W. Aigner, S. Miksch, B. Thurnher, and S. Biffl. Planninglines: Novel glyphs for representing temporal uncertainties and their evaluation. In *Proceedings of the Ninth International Conference on Information Visualisation*, IV '05, pages 457–463, Washington, DC, USA, 2005. IEEE Computer Society. ISBN 0-7695-2397-8. doi: 10.1109/IV.2005.97. URL <http://dx.doi.org/10.1109/IV.2005.97>.
- W. Aigner, S. Miksch, W. Müller, H. Schumann, and C. Tominski. Visualizing time-oriented data-a systematic view. *Comput. Graph.*, 31(3):401–409, June 2007. ISSN 0097-8493. doi: 10.1016/j.cag.2007.01.030. URL <http://dx.doi.org/10.1016/j.cag.2007.01.030>.
- N. Andrienko and G. Andrienko. *Exploratory Analysis of Spatial and Temporal Data: A Systematic Approach*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005. ISBN 3540259945.
- J. Bertin. *Semiology of Graphics: Diagrams, Networks, Maps*. Esri Press, Redlands, 2011. ISBN 978-1-589-48261-6.
- J. Bezanson, S. Karpinski, V. B. Shah, and A. Edelman. Julia: A fast dynamic language for technical computing. *CoRR*, abs/1209.5145, 2012. URL <http://dblp.uni-trier.de/db/journals/corr/corr1209.html#abs-1209-5145>.
- C. L. Borgman. Why are online catalogs hard to use? lessons learned from information-retrieval studies. *Journal of the American Society for Information Science*, 37(6):387–400, 1986. ISSN 1097-4571. doi: 10.1002/(SICI)1097-4571(198611)37:6<387::AID-ASI3>3.0.CO;2-8. URL [http://dx.doi.org/10.1002/\(SICI\)1097-4571\(198611\)37:6<387::AID-ASI3>3.0.CO;2-8](http://dx.doi.org/10.1002/(SICI)1097-4571(198611)37:6<387::AID-ASI3>3.0.CO;2-8).
- M. Bostock. bl.ocks.org, 2012. URL <http://bl.ocks.org/mbostock>. 23.07.2014.
- L. Byron and M. Wattenberg. Stacked graphs - geometry & aesthetics. *IEEE Transactions on Visualization and Computer Graphics*, 14(6):1245–1252, 2008. ISSN 1077-2626. doi: <http://doi.ieeecomputersociety.org/10.1109/TVCG.2008.166>.

## Literatur

---

- S. K. Card, J. D. Mackinlay, and B. Shneiderman, editors. *Readings in Information Visualization: Using Vision to Think*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1999. ISBN 1-55860-533-9.
- Catlett. Mega induction: Machine learning on vary large databases. In *Proceedings of the Eighth International Workshop on Machine Learning*, pages 596–599. Morgan Kaufmann, 1991.
- K. Chodorow and M. Dirolf. *MongoDB - The Definitive Guide: Powerful and Scalable Data Storage*. O'Reilly, 2010. ISBN 978-1-449-38156-1.
- A. Cockburn, A. Karlson, and B. B. Bederson. A review of overview+detail, zooming, and focus+context interfaces. *ACM Comput. Surv.*, 41(1):2:1–2:31, Jan. 2009. ISSN 0360-0300. doi: 10.1145/1456650.1456652. URL <http://doi.acm.org/10.1145/1456650.1456652>.
- J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, Jan. 2008. ISSN 0001-0782. doi: 10.1145/1327452.1327492. URL <http://doi.acm.org/10.1145/1327452.1327492>.
- S. Edlich, A. Friedland, J. Hampe, and B. Brauer. *NoSQL: Einstieg in die Welt nichtrelationaler Web 2.0 Datenbanken*. Hanser Fachbuchverlag, 10 2010. ISBN 9783446423558. URL <http://amazon.de/o/ASIN/3446423559>.
- S. G. Eick. Data visualization sliders. In *Proceedings of the 7th Annual ACM Symposium on User Interface Software and Technology*, UIST '94, pages 119–120, New York, NY, USA, 1994. ACM. ISBN 0-89791-657-3. doi: 10.1145/192426.192472. URL <http://doi.acm.org/10.1145/192426.192472>.
- G. Ellis and A. Dix. A taxonomy of clutter reduction for information visualisation. *Visualization and Computer Graphics, IEEE Transactions on*, 13(6):1216–1223, Nov 2007. ISSN 1077-2626. doi: 10.1109/TVCG.2007.70535.
- F. Fischer, F. Mansmann, and D. A. Keim. Real-time visual analytics for event data streams. In *Proceedings of the 27th Annual ACM Symposium on Applied Computing*, SAC '12, pages 801–806, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-0857-1. doi: 10.1145/2245276.2245432. URL <http://doi.acm.org/10.1145/2245276.2245432>.
- P. J. Fitzpatrick. Leading british statisticians of the nineteenth century. *Journal of the American Statistical Association*, 55(289):38–70, 1960. doi: 10.1080/01621459.1960.

10482048. URL <http://amstat.tandfonline.com/doi/abs/10.1080/01621459.1960.10482048>.
- A. Frank. Different types of 'times' in gis. In *Spatial and Temporal Reasoning in GIS*, pages 40–62. Oxford University Press, 1998.
- J. H. Friedman. Data mining and statistics: What's the connection?
- L. Golab and M. T. Özsu. Issues in data stream management. *SIGMOD Rec.*, 32(2):5–14, June 2003. ISSN 0163-5808. doi: 10.1145/776985.776986. URL <http://doi.acm.org/10.1145/776985.776986>.
- D. Goren-Bar, Y. Shahar, M. Galperin-Aizenberg, D. Boaz, and G. Tahan. Knave ii: The definition and implementation of an intelligent tool for visualization and exploration of time-oriented clinical data. In *Proceedings of the Working Conference on Advanced Visual Interfaces*, AVI '04, pages 171–174, New York, NY, USA, 2004. ACM. ISBN 1-58113-867-9. doi: 10.1145/989863.989889. URL <http://doi.acm.org/10.1145/989863.989889>.
- J. Han. *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2011. ISBN 0123814790.
- R. L. Harris. *Information Graphics: A Comprehensive Illustrated Reference*. Oxford University Press, New York, 1999. ISBN 978-0-19-513532-9.
- M. Hausenblas and J. Nadeau. *Big Data - Apache Drill: Interactive Ad-Hoc Analysis at Scale*, pages 100–104. 2013.
- S. Havre, E. Hetzler, P. Whitney, and L. Nowell. Themeriver: Visualizing thematic changes in large document collections. *IEEE Transactions on Visualization and Computer Graphics*, 8(1):9–20, Jan. 2002. ISSN 1077-2626. doi: 10.1109/2945.981848. URL <http://dx.doi.org/10.1109/2945.981848>.
- J. Heer and B. Shneiderman. Interactive dynamics for visual analysis. *Commun. ACM*, 55(4):45–54, Apr. 2012. ISSN 0001-0782. doi: 10.1145/2133806.2133821. URL <http://doi.acm.org/10.1145/2133806.2133821>.
- J. Heer, N. Kong, and M. 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>.

## Literatur

---

- J. Heer, M. Bostock, and V. Ogievetsky. A tour through the visualization zoo. *Commun. ACM*, 53(6):59–67, June 2010. ISSN 0001-0782. doi: 10.1145/1743546.1743567. URL <http://doi.acm.org/10.1145/1743546.1743567>.
- H. Hochheiser and B. Shneiderman. Dynamic query tools for time series data sets: Timebox widgets for interactive exploration. *Information Visualization*, 3(1):1–18, Mar. 2004. ISSN 1473-8716. doi: 10.1145/993176.993177. URL <http://dx.doi.org/10.1145/993176.993177>.
- C. Holz and S. Feiner. Relaxed selection techniques for querying time-series graphs. In *Proceedings of the 22Nd Annual ACM Symposium on User Interface Software and Technology*, UIST ’09, pages 213–222, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-745-5. doi: 10.1145/1622176.1622217. URL <http://doi.acm.org/10.1145/1622176.1622217>.
- R. Ihaka and R. Gentleman. R: A language for data analysis and graphics. *Journal of Computational and Graphical Statistics*, 5(3):299–314, 1996. doi: 10.1080/10618600.1996.10474713. URL <http://www.tandfonline.com/doi/abs/10.1080/10618600.1996.10474713>.
- M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly. Dryad: Distributed data-parallel programs from sequential building blocks. In *Proceedings of the 2Nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007*, EuroSys ’07, pages 59–72, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-636-3. doi: 10.1145/1272996.1273005. URL <http://doi.acm.org/10.1145/1272996.1273005>.
- W. Javed, B. McDonnel, and N. Elmquist. Graphical perception of multiple time series. *Visualization and Computer Graphics, IEEE Transactions on*, 16(6):927–934, Nov 2010. ISSN 1077-2626. doi: 10.1109/TVCG.2010.162.
- J. R. Jay Kreps, Neha Narkhede. Kafka: a distributed messaging system for log processing. NetDB workshop, 2011. URL <http://research.microsoft.com/en-us/um/people/srikanth/netdb11/netdb11papers/netdb11-final12.pdf>.
- D. Keim, H.-P. Kriegel, and M. Ankerst. Recursive pattern: a technique for visualizing very large amounts of data. In *Visualization, 1995. Visualization ’95. Proceedings., IEEE Conference on*, pages 279–286, 463, Oct 1995. doi: 10.1109/VISUAL.1995.485140.
- D. A. Keim. Pixel-oriented visualization techniques for exploring very large databases. *Journal of Computational and Graphical Statistics*, 5:58–77, 1996.

- D. A. Keim. Information visualization and visual data mining. *IEEE Transactions on Visualization and Computer Graphics*, 8(1):1–8, Jan. 2002. ISSN 1077-2626. doi: 10.1109/2945.981847. URL <http://dx.doi.org/10.1109/2945.981847>.
- R. Kincaid and H. Lam. Line graph explorer: Scalable display of line graphs using focus+context. In *Proceedings of the Working Conference on Advanced Visual Interfaces*, AVI ’06, pages 404–411, New York, NY, USA, 2006. ACM. ISBN 1-59593-353-0. doi: 10.1145/1133265.1133348. URL <http://doi.acm.org/10.1145/1133265.1133348>.
- M. Kornacker and J. Erickson. Cloudera impala: Real-time queries in apache hadoop, for real, 2012. URL <http://blog.cloudera.com/blog/2012/10/cloudera-impala-real-time-queries-in-apache-hadoop-for-real/>. 23.07.2014.
- P. Kostkova, S. Garbin, J. Moser, and W. Pan. Integration and visualization public health dashboard: The medi+board pilot project. In *Proceedings of the Companion Publication of the 23rd International Conference on World Wide Web Companion*, WWW Companion ’14, pages 657–662, Republic and Canton of Geneva, Switzerland, 2014. International World Wide Web Conferences Steering Committee. ISBN 978-1-4503-2745-9. doi: 10.1145/2567948.2579276. URL <http://dx.doi.org/10.1145/2567948.2579276>.
- M. Krstajic and D. Keim. Visualization of streaming data: Observing change and context in information visualization techniques. In *Big Data, 2013 IEEE International Conference on*, pages 41–47, Oct 2013. doi: 10.1109/BigData.2013.6691713.
- R. Lübke, R. Lungwitz, D. Schuster, and A. Schill. Large-scale tests of distributed systems with integrated emulation of advanced network behavior. *IADIS International Journal on WWW/Internet*, Vol. 10, No. 2, pp. 138-151, 0 2013.
- R. Lübke, D. Schuster, and A. Schill. Nessee: An in-house test platform for large scale tests of multimedia applications including network behavior. In *9th International Conference on Testbeds and Research Infrastructures for the Development of Networks & Communities (Tridentcom)*, Guangzhou, China, 5 2014.
- N. Leavitt. Bringing big analytics to the masses. *Computer*, 46(1):20–23, Jan 2013. ISSN 0018-9162. doi: 10.1109/MC.2013.9.
- Y. K. Leung and M. D. Apperley. A review and taxonomy of distortion-oriented presentation techniques. *ACM Trans. Comput.-Hum. Interact.*, 1(2):126–160, June 1994.

## Literatur

---

- ISSN 1073-0516. doi: 10.1145/180171.180173. URL <http://doi.acm.org/10.1145/180171.180173>.
- A. M. MacEachren. *How Maps Work - Representation, Visualization, and Design*. Guilford Press, 2004. ISBN 978-1-57230-040-8.
- J. D. Mackinlay, G. G. Robertson, and S. K. Card. The perspective wall: Detail and context smoothly integrated. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '91, pages 173–176, New York, NY, USA, 1991. ACM. ISBN 0-89791-383-3. doi: 10.1145/108844.108870. URL <http://doi.acm.org/10.1145/108844.108870>.
- S. Melnik, A. Gubarev, J. J. Long, G. Romer, S. Shivakumar, M. Tolton, and T. Vasilakis. Dremel: Interactive analysis of web-scale datasets. In *Proc. of the 36th Int'l Conf on Very Large Data Bases*, pages 330–339, 2010. URL <http://www.vldb2010.org/accept.htm>.
- D. Mintz, T. Fitz-Simons, and M. Wayland. Tracking air quality trends with sas/graph. In *SUGI 22 Proceedings*, pages 807–812, 1997.
- A. V. Moere. Time-varying data visualization using information flocking boids. In *Proceedings of the IEEE Symposium on Information Visualization*, INFOVIS '04, pages 97–104, Washington, DC, USA, 2004. IEEE Computer Society. ISBN 0-7803-8779-3. doi: 10.1109/INFOVIS.2004.65. URL <http://dx.doi.org/10.1109/INFOVIS.2004.65>.
- C. Plaisant. The challenge of information visualization evaluation. In *Proceedings of the Working Conference on Advanced Visual Interfaces*, AVI '04, pages 109–116, New York, NY, USA, 2004. ACM. ISBN 1-58113-867-9. doi: 10.1145/989863.989880. URL <http://doi.acm.org/10.1145/989863.989880>.
- W. Playfair. *The commercial and political atlas: Representing, by means of stained copper-plate charts, the progress of the commerce, revenues, expenditure and debts of England during the whole of the Eighteenth Century*. London, 1786.
- G. Proctor and C. Winter. Information flocking: Data visualisation in virtual worlds using emergent behaviours. In *In Virtual Worlds 98*, pages 168–176. Springer-Verlag, 1998.
- H. Reijner and P. Software. The development of the horizon graph, 2008.

- G. Roden. *Node.js & Co.: Skalierbare, hochperformante und echtzeitfähige Webanwendungen in JavaScript entwickeln*. Dpunkt.Verlag GmbH, 2014. ISBN 9783864901324. URL <http://books.google.de/books?id=a8QXngEACAAJ>.
- M. Rueter and E. Fields. Tableau for the enterprise: An overview for it, 2012. URL [http://www.tableausoftware.com/sites/default/files/whitepapers/whitepaper\\_tableau-for-the-enterprise\\_0.pdf](http://www.tableausoftware.com/sites/default/files/whitepapers/whitepaper_tableau-for-the-enterprise_0.pdf).
- T. Saito, H. N. Miyamura, M. Yamamoto, H. Saito, Y. Hoshiya, and T. Kaseda. Two-tone pseudo coloring: Compact visualization for one-dimensional data. *Information Visualization, IEEE Symposium on*, 0:23, 2005. ISSN 1522-404x. doi: <http://doi.ieeecomputersociety.org/10.1109/INFOVIS.2005.35>.
- M. Sarkar and M. H. Brown. Graphical fisheye views. *Commun. ACM*, 37(12):73–83, Dec. 1994. ISSN 0001-0782. doi: 10.1145/198366.198384. URL <http://doi.acm.org/10.1145/198366.198384>.
- A. Shabtai, D. Klimov, Y. Shahar, and Y. Elovici. An intelligent, interactive tool for exploration and visualization of time-oriented security data. In *Proceedings of the 3rd International Workshop on Visualization for Computer Security*, VizSEC ’06, pages 15–22, New York, NY, USA, 2006. ACM. ISBN 1-59593-549-5. doi: 10.1145/1179576.1179580. URL <http://doi.acm.org/10.1145/1179576.1179580>.
- B. Schneiderman. The eyes have it: A task by data type taxonomy for information visualizations. In *Proceedings of the 1996 IEEE Symposium on Visual Languages*, VL ’96, pages 336–, Washington, DC, USA, 1996. IEEE Computer Society. ISBN 0-8186-7508-X. URL <http://dl.acm.org/citation.cfm?id=832277.834354>.
- B. Schneiderman. *Designing the User Interface – Strategies for Effective Human-Computer-Interaction*. Addison-Wesley Longman, 3 edition, 1998. ISBN 0201694972.
- K. Shvachko, H. Kuang, S. Radia, and R. Chansler. The hadoop distributed file system. In *Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on*, pages 1–10, May 2010. doi: 10.1109/MSST.2010.5496972.
- S. Sorkin. Splunk technical paper: Large-scale, unstructured data retrieval and analysis using splunk., 2011. URL [http://de.splunk.com/web\\_assets/pdfs/secure/Splunk\\_and\\_MapReduce.pdf](http://de.splunk.com/web_assets/pdfs/secure/Splunk_and_MapReduce.pdf).

## Literatur

---

- W. Stier. *Methoden Der Zeitreihenanalyse*. Springer-Lehrbuch. Springer Berlin Heidelberg, 2001. ISBN 9783540417002. URL <http://books.google.de/books?id=J9INZfyZSw8C>.
- F. Stoffel, F. Fischer, and D. A. Keim. Finding anomalies in time-series using visual correlation for interactive root cause analysis. In *Proceedings of the Tenth Workshop on Visualization for Cyber Security*, VizSec '13, pages 65–72, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-2173-0. doi: 10.1145/2517957.2517966. URL <http://doi.acm.org/10.1145/2517957.2517966>.
- Storm. URL <https://storm.incubator.apache.org/>. 22.07.2014.
- C. Tominski, J. Abello, and H. Schumann. Axes-based visualizations with radial layouts. In *Proceedings of the 2004 ACM Symposium on Applied Computing*, SAC '04, pages 1242–1247, New York, NY, USA, 2004. ACM. ISBN 1-58113-812-1. doi: 10.1145/967900.968153. URL <http://doi.acm.org/10.1145/967900.968153>.
- E. R. Tufte. *The Visual Display of Quantitative Information*. Graphics Press, 1983.
- J. van Wijk and E. Van Selow. Cluster and calendar based visualization of time series data. In *Information Visualization, 1999. (Info Vis '99) Proceedings. 1999 IEEE Symposium on*, pages 4–9, 140, 1999. doi: 10.1109/INFVIS.1999.801851.
- J. J. Van Wijk and W. A. A. Nuij. Smooth and efficient zooming and panning. In *Proceedings of the Ninth Annual IEEE Conference on Information Visualization*, INFOVIS'03, pages 15–22, Washington, DC, USA, 2003. IEEE Computer Society. ISBN 0-7803-8154-8. URL <http://dl.acm.org/citation.cfm?id=1947368.1947376>.
- M. Weber, M. Alexa, and W. Müller. Visualizing time-series on spirals. In *Proceedings of the IEEE Symposium on Information Visualization 2001 (INFOVIS'01)*, INFOVIS '01, pages 7–, Washington, DC, USA, 2001. IEEE Computer Society. ISBN 0-7695-1342-5. URL <http://dl.acm.org/citation.cfm?id=580582.857719>.
- C. Williamson and B. Shneiderman. The dynamic homefinder: Evaluating dynamic queries in a real-estate information exploration system. In *Proceedings of the 15th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '92, pages 338–346, New York, NY, USA, 1992. ACM. ISBN 0-89791-523-2. doi: 10.1145/133160.133216. URL <http://doi.acm.org/10.1145/133160.133216>.

- Y.-L. Wu, D. Agrawal, and A. El Abbadi. A comparison of dft and dwt based similarity search in time-series databases. In *Proceedings of the Ninth International Conference on Information and Knowledge Management*, CIKM '00, pages 488–495, New York, NY, USA, 2000. ACM. ISBN 1-58113-320-0. doi: 10.1145/354756.354857. URL <http://doi.acm.org/10.1145/354756.354857>.
- G. U. Yule. On a method of investigating periodicities in disturbed series, with special reference to wolfer's sunspot numbers. *Philosophical Transactions of the Royal Society of London. Series A, Containing Papers of a Mathematical or Physical Character*, 226(636-646):267–298, 1927. doi: 10.1098/rsta.1927.0007. URL <http://rsta.royalsocietypublishing.org/content/226/636-646/267.short>.
- M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica. Spark: Cluster computing with working sets. In *Proceedings of the 2Nd USENIX Conference on Hot Topics in Cloud Computing*, HotCloud'10, pages 10–10, Berkeley, CA, USA, 2010. USENIX Association. URL <http://dl.acm.org/citation.cfm?id=1863103.1863113>.