WWU
MÜNSTER

The Goal-oriented Business Intelligence Architectures Method

A Process-based Approach to Combine Traditional and Novel Analytical Technologies

David Fekete

David Fekete

The Goal-oriented Business Intelligence Architectures Method: A Process-based Approach to Combine Traditional and Novel Analytical Technologies

Inauguraldissertation zur Erlangung des akademischen Grades eines Doktors der Wirtschaftswissenschaften durch die Wirtschaftswissenschaftliche Fakultät der Westfälischen Wilhelms-Universität Münster

Vorgelegt von

David Fekete, M.Sc.
aus Szolnok, Ungarn

Mai 2018

Dekanin	Prof. Dr. Theresia Theurl
Erster Gutachter	Prof. Dr. Gottfried Vossen
Zweiter Gutachter	Prof. Dr. Dr. h.c. Dr. h.c. Jörg Becker
Dritter Gutachter	Prof. Dr. Jens Leker
Mündliche Prüfung	06.07.2018

David Fekete

The Goal-oriented Business Intelligence Architectures Method

Wissenschaftliche Schriften der WWU Münster

Reihe IV

Band 18

David Fekete

The Goal-oriented Business Intelligence Architectures Method

A Process-based Approach to Combine Traditional and Novel Analytical Technologies

Wissenschaftliche Schriften der WWU Münster
herausgegeben von der Universitäts- und Landesbibliothek Münster
http://www.ulb.uni-muenster.de

Bibliografische Information der Deutschen Nationalbibliothek:
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über https://www.dnb.de abrufbar.

Dieses Buch steht gleichzeitig in einer elektronischen Version über den Publikations- und Archivierungsserver der WWU Münster zur Verfügung.
https://www.ulb.uni-muenster.de/wissenschaftliche-schriften

David Fekete
„The Goal-oriented Business Intelligence Architectures Method. A Process-based Approach to Combine Traditional and Novel Analytical Technologies“
Wissenschaftliche Schriften der WWU Münster, Reihe IV, Band 18
Verlag readbox unipress in der readbox publishing GmbH, Dortmund
www.readbox.net/unipress

Zugl.: Diss. Universität Münster, 2018

ISBN 978-3-8405-0226-2 (Druckausgabe)
URN urn:nbn:de:hbz:6-11169662922 (elektronische Version)

direkt zur Online-Version:

Satz: David Fekete
Titelbild: David Fekete
Umschlag: ULB Münster

Geleitwort

In den letzten 10 Jahren ist der Begriff „Big Data“ von einem eher technischen Begriff zu einem Schlagwort mutiert, das positive wie negative Effekte des durch die umfassende Digitalisierung aller Branchen und Lebensbereiche sowie durch den Preisverfall bei Computer-Hardware ermöglichten Datensammelns subsummiert. War IT-Technik früher teuer, so dass man sich für geplante Datenauswertungen genau überlegen musste, welche Daten wie zu erheben waren, wird heute jeder Klick im Internet, jede Nachricht (auch zwischen technischen Systemen) und jedes digitale Signal irgendwo gespeichert in der Hoffnung, irgendwann aus seiner Auswertung gewinnbringende Schlüsse ziehen zu können. Die Speerspitze dieser Bewegung bilden einerseits soziale Netzwerke und andererseits der E-Commerce, der nach wie vor zweistellige Zuwachsraten pro Jahr aufweist. Allerdings waren die letzten Jahre nicht nur durch einen Verfall der Hardware-Preise gekennzeichnet (dies ist immerhin seit rund 50 Jahren so), sondern auch durch dramatische Veränderungen im Bereich Software: Einerseits haben gerade die Protagonisten des Big Data (wie Google, Twitter oder Facebook) das Erstellen von Software etwa durch Eigenentwicklung von Programmiersprachen oder Frameworks vereinfacht; andererseits hat sich die Bereitstellung zuverlässiger und flexibler Software-Werkzeuge für alle denkbaren Aspekte des Umgangs mit und der Verarbeitung von Big Data durch die Open-Source-Bewegung und eine Bereitstellung als Cloud-Lösungen dramatisch beschleunigt. Fast täglich ist in einschlägigen Blogs von neuen Werkzeugen zu lesen, die entweder alte ersetzen, weiterentwickeln oder ergänzen; nicht selten kommt man zu deren Anwendung mit wenig Programmierung aus, so dass man sich im Wesentlichen auf eine angemessene Konfigurierung beschränken kann.

In dieser Situation stellt sich die Frage, wie man als Anwender vorgehen soll, wenn man eine bestimmte Aufgabe im Big Data-Kontext einerseits

technisch lösen soll, andererseits aber den vorhandenen Unternehmens-IT-Kontext nicht aus den Augen verlieren darf. Für viele Unternehmen erhält diese Frage ihre Brisanz dadurch, dass man in den letzten 20 Jahren teilweise massiv in Technologie für „Business Intelligence“ (BI) und insbesondere Data Warehouses investiert hat und daher keineswegs an einer radikalen Ablösung, sondern eher an einer schrittweisen Migration oder einer Erweiterung der vorhandenen Technik interessiert ist. Diesen Fragen widmet sich die Dissertation von David Fekete; er präsentiert die von ihm entwickelte GOBIA-Methode (Goal-oriented Business Intelligence Architectures method), bei der es sich um eine prozessorientierte Vorgehensweise zu Entwicklung von Big Data-Anwendungen handelt, die einerseits von konkreten Fragestellungen und den daraus resultierenden Anforderungen ausgeht und die andererseits sowohl vorhandene BI-Lösungen als auch neuere Software-Entwicklungen in die Betrachtungen einbezieht.

David Fekete legt mit seiner Dissertation ein sehr umfassendes und gelungenes Werk vor. Es weist ihn als profunden Kenner der Big Data-Materie aus, der in der Lage ist, sein tiefes Technik-Verständnis mit Informatik-Techniken wie Requirements Engineering oder Prozessmodellierung so zu kombinieren, dass sich daraus eine angemessene Methode zur Entwicklung von Big Data-Architekturen ergibt. Er ist vollkommen sicher im Umgang mit Systemarchitekturen, kann diese einordnen und kritisch bewerten und kann ferner daraus seine eigene Methodik ableiten, die er angemessen in die Phasen „REF“ und „DEV“ unterteilt. Die von ihm entwickelte Methode zum Entwurf und zur Realisierung einer Big Data-Architektur ist in ihrer Darstellung und Durchführung neu, wenngleich sie naturgemäß auf einschlägigen Vorarbeiten basiert, die über Jahre hinweg durchgeführt und auch in Praxisprojekten erprobt wurden. Genau diese Vorarbeiten kann er allerdings geschickt als Use Cases verarbeiten, die er dazu verwendet, die Anwendung seiner GOBIA-Methode zu illustrieren. Die Arbeit stellt insgesamt eine wichtige Ergänzung zum Vorgehen bei und Wissenstand um Big Data-Applikationen dar; ich wünsche ihr daher eine breite Leserschaft.

Münster, April 2020 Prof. Dr. Gottfried Vossen

Acknowledgements

First and foremost, I would like to express my profound gratitude to my doctoral adviser Prof. Dr. Gottfried Vossen for giving me the opportunity to conduct research in his DBIS Group in Münster along with his relentless guidance during my research and while writing this thesis. I am grateful for his trust in my capabilities to contribute to the vast fields of Business Intelligence and Data Warehousing, which gain new opportunities through Big Data. He encouraged me and supported my journey into these topics since my Bachelor thesis. I am thankful for our discussions and his advice on my research. I would also like to extend my gratitude to Prof. Dr. Jörg Becker for engaging as my second reviewer as well as his helpful feedback and also to Prof. Dr. Jens Leker for assuming the role of the third reviewer for the defense of this thesis.

My colleagues from the DBIS Group have supported me greatly during my research and the writing process of this thesis and I am thankful for their various contributions. I would like to thank — in no particular order — my long-time office mate Dr. Nicolas Pflanzl for challenging my thought processes for the thesis and proof-reading parts of it, Fabian Schomm-von Auenmüller for our insightful exchanges during the final phase of our respective writing processes, Dr. Jens Lechtenbörger for his precise observations and ideas especially during the joint discussions of my results, Ralf Farke for his excellent technical support and our team assistants through the years Claudia Werkmeister, Lena Hertzel, Katharina Hoffmann, Melanie Wietkamp and their successor Julia Seither. I also thank Dr. Florian Stahl, Leschek Homann, Denis Martins and Felix Nolte who provided valuable feedback during my research.

I am deeply grateful to a dear friend of mine, Dr. Vincent von Hof, who not just flexibly invested at lot of time to proof-read almost the entire thesis but

was continuously willing to exchange ideas on writing and methodological aspects.

My parents and my older brother gave me their wholehearted and unwavering encouragement during my doctorate. I could not be more thankful for such backing that allowed me to focus on my research. My eternal gratitude goes to my girlfriend Sandra, the love of my life, who patiently stood by my side and supported me not only with her warmth, heart and kindness but also with proof-reading excerpts of my doctoral thesis — even though being in the process of writing her own. She gave me both the comfort and strength to always push forward and successfully conclude this journey.

Arnsberg, April 2020 David Fekete

Contents

1 Introduction 1
- 1.1 Motivation 1
- 1.2 Structure and Research Method 6
- 1.3 FROG AIR Sample Case 15

2 Background 17
- 2.1 Traditional Business Intelligence 17
  - 2.1.1 Relational Database Management Systems and SQL . 21
  - 2.1.2 Data Warehouses 31
- 2.2 Big Data Technologies 50
  - 2.2.1 Big Data Phenomenon and Characteristics 50
  - 2.2.2 NoSQL Data Stores 61
  - 2.2.3 Hadoop Distributed File System (HDFS) 72
  - 2.2.4 MapReduce 79
  - 2.2.5 Hadoop Ecosystem 86
  - 2.2.6 Stream Processing 89
  - 2.2.7 Advancements to Traditional Technologies 107
- 2.3 Big Data Analytics 110
  - 2.3.1 Analytics in the Big Data Age 110
  - 2.3.2 Machine Learning 114
  - 2.3.3 Methods 118
  - 2.3.4 New Analytics Domains 124
- 2.4 Process Modeling 129
- 2.5 Requirements Engineering 132
  - 2.5.1 Goals 136
  - 2.5.2 Requirements 137
  - 2.5.3 System Development Approaches 141

2.5.4 FROG AIR Case Example 142
2.6 Multi-Perspective Architecture Analysis 149
2.6.1 Methods for Strategic Analysis 149
2.6.2 TORE Analysis Framework 152

3 Analytical Architectures in Research and Practice 155
3.1 Architectures Background 156
3.2 Big Data Value Chains 164
3.2.1 Analysis of Existing Value Chains 165
3.2.2 A Unified Big Data Value Chain 173
3.3 Data Warehouse Architecture Advancements 178
3.3.1 Data Warehouse 2.0 by Inmon 179
3.3.2 Data Warehouses and Big Data 180
3.4 Big Data Reference Architectures 183
3.4.1 Big Data Reference Architecture by Pääkkönen and Pakkala 184
3.4.2 NIST Big Data Reference Architecture 189
3.4.3 Big Data Solution Reference Architecture by Geerdink 197
3.4.4 Oracle Big Data Platform 201
3.4.5 Lambda Architecture by Marz and Warren 206
3.4.6 SOLID Architecture by Martínez-Prieto et al. . . 211
3.4.7 Analysis 214
3.5 Big Data Architectures in Practice 222
3.6 Selection Criteria for Analytical Architectures 230
3.7 Solution Artifact Requirements 234
3.7.1 Solution and Design Principles 235
3.7.2 Requirements Overview 238

4 Goal-oriented Business Intelligence Architectures (GOBIA) Method 241
4.1 Overview of the Approach 242
4.1.1 Design Approach 244
4.1.2 Method Scope 247

4.1.3 Execution in Cycles . . . 252
4.2 GOBIA.REF . . . 253
4.2.1 Functional Reference Architecture . . . 254
4.2.2 Technological Reference Architecture . . . 259
4.3 GOBIA.DEV . . . 272
4.3.1 Phase I . . . 273
4.3.2 Phase II . . . 288
4.3.3 Phase III . . . 298
4.4 Post-GOBIA Activities . . . 318
4.5 Extending the GOBIA Method . . . 322
4.5.1 Updating the GOBIA Tool Repository . . . 323
4.5.2 Updating GOBIA.REF . . . 331
4.5.3 Updating GOBIA.DEV . . . 333

5 Evaluation . . . 337
5.1 Evaluation Method . . . 337
5.2 Selected Case Applications . . . 339
5.2.1 Stream Analytics: Hospital Asset Management . . . 340
5.2.2 Advanced Analytics: Housing Price Prediction . . . 368
5.2.3 Big Data Analytics: Plagiarism Checking . . . 383
5.3 Evaluation Results Discussion . . . 397
5.3.1 Summary of Results . . . 397
5.3.2 Reflection on the Solution Artifact Requirements . . . 400

6 Discussion . . . 407

7 Conclusion . . . 411
7.1 Summary . . . 411
7.2 Outlook . . . 415

Bibliography . . . 419

List of Web Pages . . . 447

List of Abbreviations 485

A Architecture Literature Search Details 491

B Supplementary GOBIA Models 497

C GOBIA Tool Repository 507
C.1 Tool Lists 507
C.1.1 Data Acquisition 508
C.1.2 Data Storage 509
C.1.3 Data Processing 513
C.1.4 Data Analytics 513
C.2 Compatibility Matrix 519

List of Figures

1.1 Process of Design Science research 11
1.2 Work structure and steps of Design Science method 13

2.1 DBS usage scenario with DBMS, databases, and users 22
2.2 OLAP cube example 42
2.3 DWH Snowflake and Star Schema 43
2.4 DWH reference architecture 46
2.5 HDFS Architecture 76
2.6 Hadoop MapReduce data flow 80
2.7 Cycle of data processing, analysis, and storage as Petri net . . . 131
2.8 Illustration of refinements in Petri nets 131
2.9 Petri net notations used in this work 133
2.10 Three requirements engineering artifacts and their relationships 135

3.1 Beauvais Cathedral in Picardy, France 157
3.2 Conceptual relationships between reference architectures, reference models, architectural patterns, and concrete architectures . 161
3.3 Categorization of Big Data Value Chains 166
3.4 Big Data Value Chain for Streaming Applications based on [131] 167
3.5 Data Value Chain based on [163] 168
3.6 Big Data Value Chain as both in [HWCL14] and [CML14] 170
3.7 Big Data Value Chain based on [Cur16] 172
3.8 Unified Big Data Value Chain 174
3.9 Data Warehouse Reference Architecture augmented with Big Data by Shweta and Nandi 182
3.10 Big Data reference architecture by Pääkkönen and Pakkala . . 186
3.11 NIST Big Data Reference Architecture 196

3.12 Big Data Solution Reference Architecture by Geerdink 199
3.13 Oracle Big Data Platform 203
3.14 Oracle Data and Technology Map 205
3.15 Lambda Architecture by Marz and Warren 208
3.16 SOLID Architecture by Martínez-Prieto et al. 212
3.17 Architectural Pattern used at Netflix [Ama13] 223
3.18 Architectural Pattern used at LinkedIn [SKG+12] 225
3.19 Architectural Pattern used at Twitter [MDL+13] 228

4.1 GOBIA method in its specific and general scope in a broader organizational context 251
4.2 GOBIA.REF Technological Reference Architecture 255
4.3 GOBIA.REF Technological Reference Architecture 261
4.4 Conceptual illustration of GOBIA.DEV execution 274
4.5 GOBIA.DEV, Phase I Process Model (Short Version) 282
4.6 GOBIA.DEV Phase I: FROG AIR execution 1/2 283
4.7 GOBIA.DEV Phase I: FROG AIR execution 2/2 286
4.8 GOBIA.DEV Phase II Process Model 290
4.9 GOBIA.DEV: FROG AIR abstract BI architecture 295
4.10 GOBIA.DEV Phase II: FROG AIR excerpt of semi-concrete BI architectures as tactical plan overview 296
4.11 GOBIA.DEV Phase II: FROG AIR tactical plan TP_1 299
4.12 GOBIA.DEV Phase III Process Model (Short Version) 302
4.13 GOBIA.DEV Phase III Process Model: Selection of Data Acquisition Technology (Extract) 304
4.14 GOBIA.DEV Phase III Process Model: Selection of Data Storage Technology (Extract) 306
4.15 GOBIA.DEV Phase III Process Model: Selection of Data Processing Technology (Extract) 309
4.16 GOBIA.DEV Phase III Process Model: Selection of Data Analytics Technology (Extract) 310
4.17 GOBIA.DEV Phase III: FROG AIR simplified compilation of BI architecture alternatives in strict layers 319
4.18 The evaluation of alternatives as post-GOBIA Process 320

5.1 Example for a heat-map laid upon on a floor plan 345
5.2 GOBIA.DEV Phase I: HPL execution 1/2 353
5.3 GOBIA.DEV Phase I: HPL execution 2/2 355
5.4 GOBIA.DEV: HPL abstract BI architecture 357
5.5 GOBIA.DEV Phase II: HPL excerpt of semi-concrete BI architectures as tactical plan overview 358
5.6 GOBIA.DEV Phase III: HPL case processing technology selection 362
5.7 GOBIA.DEV Phase III: HPL illustration of BI architecture alternative in strict layers 367
5.8 GOBIA.DEV Phase I: KRE execution 375
5.9 GOBIA.DEV: KRE abstract BI architecture 377
5.10 GOBIA.DEV Phase II: KRE excerpt of semi-concrete BI architectures as tactical plan overview 378
5.11 GOBIA.DEV Phase III: KRE illustration of BI architecture alternative in strict layers 383
5.12 GOBIA.DEV Phase I: Plagcheck execution 390
5.13 GOBIA.DEV: PlagCheck abstract BI architecture 393
5.14 GOBIA.DEV Phase II: PlagCheck excerpt of semi-concrete BI architectures as tactical plan overview 394
5.15 Utilization of the GOBIA.REF technological reference architecture in Case Applications 405

B.1 GOBIA.DEV, Phase I Process Model, Part 1 498
B.2 GOBIA.DEV, Phase I Process Model, Part 2 499
B.3 GOBIA.DEV, Phase III Process Model, Part 1 500
B.4 GOBIA.DEV, Phase III Process Model, Part 2 501
B.5 GOBIA.DEV, Phase II FROG AIR case: Semi-concrete BI architecture 502
B.6 GOBIA.DEV, Phase II HPL case: Semi-concrete BI architecture . 503
B.7 GOBIA.DEV, Phase II KRE case: Semi-concrete BI architecture . 504
B.8 GOBIA.DEV, Phase III: HPL simplified compilation of BI architecture alternatives in strict layers 505
B.9 GOBIA.REF Technological Reference Architecture without Coloring 506

List of Tables

2.1 Relational example table *RepairJobs* 24
2.2 Selection of exemplary Big Data definitions 56
2.3 Variety dimension of Big Data with examples 59
2.4 Example NoSQL data stores by category 65
2.5 Overview of time spans to distinguish systems by latency types 93
2.6 Properties of various Streaming Processing Technologies 106
2.7 NewSQL system examples. 109
2.8 Goal documentation tempalte 138

3.1 Big Data reference architecture analysis summary 220

4.1 Goal-oriented Business Intelligence Architectures (GOBIA) tool repository overview for the Data Acquisition layer 262
4.2 GOBIA tool repository overview for the Data Storage layer . . . 267
4.3 GOBIA tool repository for the Data Processing layer 269
4.4 GOBIA tool repository for the Data Analytics layer 272
4.5 Sample expressions and words of requirement descriptions to BI functionality 279
4.6 GOBIA.DEV Phase I: FROG AIR initial input 281
4.7 GOBIA.DEV Phase I: FROG AIR requirements separation 283
4.8 GOBIA.DEV Phase I: FROG AIR BI functionalities and BIDEs . . 285
4.9 GOBIA.DEV Phase I: New requirements 285
4.10 GOBIA.DEV Phase I: FROG AIR Data preparation steps 287
4.11 GOBIA.DEV Phase II: Tactical plans for FROG AIR 298
4.12 GOBIA.DEV Phase II: List of Architecture Building Blocks through all FROG AIR tactical plans 300
4.13 Decision criteria for technology selection in GOBIA.DEV Phase III 311

4.14 Technology classes selected for FROG AIR in GOBIA.DEV Phase III . . . 317
4.15 Adding Vertica to the GOBIA Tool Repository . . . 326
4.16 Adding a *Neural Network* capability attribute to the GOBIA Tool Repository . . . 330

5.1 GOBIA.DEV Phase I: HPL initial input . . . 352
5.2 GOBIA.DEV Phase I: HPL requirements separation . . . 352
5.3 GOBIA.DEV Phase I: HPL BI functionalities and BI data entities (BIDEs) . . . 354
5.4 GOBIA.DEV Phase I: HPL Data preparation steps . . . 356
5.5 GOBIA.DEV Phase II: Tactical plans for HPL . . . 360
5.6 GOBIA.DEV Phase II: List of Architecture Building Blocks through all HPL tactical plans . . . 360
5.7 Technology classes selected for HPL in GOBIA.DEV Phase III . 365
5.8 GOBIA.DEV Phase I: KRE initial input . . . 373
5.9 GOBIA.DEV Phase I: KRE requirements separation . . . 374
5.10 GOBIA.DEV Phase I: KRE BI functionalities and BIDEs . . . 376
5.11 GOBIA.DEV Phase I: KRE Data preparation steps . . . 377
5.12 GOBIA.DEV Phase II: Tactical plan for the KRE case . . . 379
5.13 GOBIA.DEV Phase II: List of Architecture Building Blocks for the KRE case . . . 380
5.14 Technology classes selected for KRE in GOBIA.DEV Phase III . 382
5.15 GOBIA.DEV Phase I: PlagCheck initial input . . . 388
5.16 GOBIA.DEV Phase I: PlagCheck requirements separation . . . 389
5.17 GOBIA.DEV Phase I: PlagCheck BI functionalities and BIDEs . . 391
5.18 GOBIA.DEV Phase I: PlagCheck Data preparation steps . . . 392
5.19 GOBIA.DEV Phase II: List of Architecture Building Blocks through all PlagCheck tactical plans . . . 395

A.1 Search keywords and engines for literature search on Big Data architectures . . . 492
A.2 Relevant results of the literature search on Big Data architectures . 493

A.3 Out of scope results of the literature search on Big Data architectures 494

C.1 GOBIA tool repository: Tools for Data Acquisition 510
C.2 GOBIA tool repository: Tools for Data Storage 514
C.3 GOBIA tool repository: Tools for Data Processing. 518
C.4 GOBIA tool repository: Tools for Data Analytics 520
C.5 GOBIA tool repository: Compatbility Matrix 525

1 Introduction

> *"As business leaders we need to understand that lack of data is not the issue. Most businesses have more than enough data to use constructively; we just don't know how to use it. The reality is that most businesses are already data rich, but insight poor."*
>
> Bernard Marr [Mar15, p. 19]

1.1 Motivation

Analyzing data to gain information for decision-making has been a key interest for mankind for a long time. 20,000 years ago, humans already ingrained sticks and bones with markings to create early forms of data storage, which they even used for food supply predictions [244]. The rapid development of Information Technology (IT) in the second of half of the 20th century spurred the use and analysis of data in digital form for organizational decisions. IT systems started to provide business managers with structured information such as weekly revenue reports or budget forecasts, which they used to guide their business decisions. All this was based on analyses of recorded data. At that time, the first cornerstones of what is known today as *Business Intelligence (BI)* were laid [Pow04, Luh58], whose goal is to enable business decision support using empirical data [GRM15, p. vii]. Since that time, IT-based data analysis evoked widespread interest in both academia and practice (e. g., cf. [GRM15, p. 1], [PF13, p. 1], [Pow04, 334], [LL15, p. 5]). In the 1990s, Data Warehouses (DWHs) as contemporary analysis systems

emerged and gained far-reaching attention as a technology for data analysis [Pow04, Inm96, Kim96].

Technologically, DWHs are based on Relational Database Management Systems (RDBMSs), which store data in a structured form, e. g., as tables. However, DWHs are specifically designed for complex analytical tasks such as Online Analytical Processing (OLAP) or Data Mining. RDBMSs and their predecessors are aimed at data in day-to-day business operations such as bank transactions and are thus termed Online Transaction Processing (OLTP) systems. Data Warehouses also represent a conceptually different approach. By aiming to create an integrated set of data from heterogeneously structured, original OLTP sources, a *single point of truth* that preserves the history of its source data is formed [ISN08, p. 7]. This also means that "raw" source data needs to be pre-processed and integrated before it can be used for analytical purposes.

Together with Data Warehouses, the term BI also gained popularity in the 1990s. BI systems became closely associated with the widespread and related DWHs including OLAP and Data Mining — even so far that the terms were used synonymously, in spite of BI originally being a technology-agnostic approach to decision support (e. g., cf. [GRM15]). In 1992, Walmart used its Data Warehouse to store and disseminate sales data in under 10 minutes at a central location, which quickly grew to more than 1 TB [356]. This was an impressive amount of data at that time. As such Data Warehouse systems followed a common functional pattern, in time a common blueprint to build a custom Data Warehouse emerged, the so-called *DWH reference architecture* (cf. [Inm96, ISN08, KR13]). Data is extracted from its original sources, cleansed, transformed, and loaded into a separate DWH system. From there, it is distributed to BI applications such as reporting or visualization tools and other analytic software.

The advent of first *Web 2.0* and then *Big Data* in the first decade of the 21st century brought numerous challenges and opportunities to this "traditional" concept and data analysis in general [Cat11]. These new developments questioned the "one size fits all" approach, which DWHs and RDBMSs were

thought to be [SC05]. Social media sites like Facebook[1] and Twitter[2] needed a simpler and more flexible form of storage, which could cope with constantly evolving services used by millions of people [Vos14, p. 4]. Not only SQL (NoSQL) data stores addressed that by trading transactional guarantees and structured storage for better scalability, availability, and flexible data structures [Cat11]. E. g., for Twitter it is less relevant if their post is immediately visible around the world, but rather that the service reacts promptly in general and does not collapse when being used by millions of users in parallel.

Just as distributed storage, distributed processing of massive data sets became a key driver of the Big Data hype [181], which began around the same time period. This trend tapped into the dormant potential of less-structured or unstructured data such as e-mails, Web logs, images, and videos — forms of data which Warehouses and RDBMSs are not designed for [ISN08]. Simultaneously, Big Data led to the utilization of constantly increasing volumes of data around the globe. While the Walmart Warehouse with 1 TB of data was considered "big" in 1992, contemporary systems hold hundreds of TBs of data on average [Rus11, p. 16] and even vastly more in some cases [201]. Still, these systems are dwarfed by the hundreds of PBs of data sent through the Web every minute and even by individual Big Data systems which are designed to scale to enormous sizes of hundreds of PBs in total (e. g., [392]). Overall, the Big Data trend lead to many new technologies such as Apache Hadoop [103] and Apache Spark [76], as traditional technology was not able to handle such huge data volumes.

Nowadays, Big Data is the cornerstone of many innovations, improved existing use cases and has enabled analytics not possible before (cf. [Mar15, Mar16]). For instance, agricultural manufacturer John Deere[3] uses a Big Data-enabled real-time monitoring system for its customers to help them optimize their crops planning and increase their yields [Mar16, p. 75f.]. Another

[1] https://www.facebook.com.
[2] https://www.twitter.com.
[3] http://www.deere.com.

example is fashion company Ralph Lauren[4], who developed a "connected" PoloTech Shirt that allows tracking its wearer's movements, steps, heartbeat, and burnt calories [Mar16, p. 195].

Especially vendors of so-called *Hadoop distributions* provide "Big Data" capable software packages that consist of multiple novel Big Data tools for various tasks, such as data loading, streaming analysis, and distributed batch processing. For instance, *MapR* [243] and *Hortonworks* [208] include more than a dozen tools that are open-source and thus offered for free. Nevertheless, these Hadoop distributions do not constitute a commonly agreed-upon reference architecture, which would allow to select the proper set of tools for the use case at hand. Moreover, a new generation of "beyond Hadoop" [Mon13, Agn14] technologies such as Apache Spark [76] have since emerged, which focus on advanced concepts such as streaming data analytics. These tools enable real-time processing and analyses that outgrow the scope of traditional processing and analytics technologies [Agn14] [Psa17, pp. 3ff.]. For instance, analyzing live Twitter posts and visualizing the data needs a fast processing pipeline focused on speed rather than accuracy and consistency, which is diametrical to traditional offerings with heavy pre-processing pipelines designed to work on highly-valued business source data. For instance, reactions to elections in the United States on Twitter were analyzed using real-time techniques [397], which need to cope with a fast streams of data.

Despite these developments and doubts about its future utility [209, 377], DWHs remain an established concept and a valid choice for the analysis of well-structured, high-valued data and are still successfully used by many organizations [Rus11]. Nevertheless, DWHs need to evolve as well, and might even incorporate some of the newer approaches [371, 170, 234]. Choosing the proper traditional or novel tools for a given problem is becoming increasingly challenging due to this abundance of choices that cover a much wider range of use cases than in the past. Besides the sheer number of tools (e. g., cf. [381]), their application patterns have changed as well, which allow for different combinations to support various use cases (e. g., [Inm16]).

[4] https://www.ralphlauren.com/.

Given the abundance of tools available and the fact that the range of possible use cases has been widened significantly, the question arises how a customized BI architecture can be developed, i. e., constructed. Such a BI architecture fulfills the original purpose of BI, which is decision support — irrespective of a particular technology. When an organization desires to solve a business problem, it needs to decide on a proper architecture to support its decision-making. For instance, in a survey by Russom, more than a third of the respondents stated that architecting an analytics system poses a challenge for their organizations, making it one of the top issues outlined [Rus11, p. 12]. Reference architectures have the purpose to enable the development of a customized architecture and provide the building blocks for doing so [AGG09, p. 3]. Therefore, a universal *BI reference architecture* is needed, which integrates both novel Big Data technologies and traditional ones as DWHs. However, depicting all these technologies and highly-varied usage patterns in form of building blocks does not yet conclusively answer *which* of these are needed for the construction of a customized BI architecture. As it has to be determined which technologies are able to solve a given problem, concise guidance to select the proper ones out of many possibilities given a specific use case is required.

Research Questions

Consequently, this work aims to answer the following research questions:

$\mathbf{RQ_1}$ How can the construction of a suitable BI architecture for a given use case be supported by a universal BI reference architecture, taking a traditional BI background into account?

$\mathbf{RQ_2}$ Which process definition enables the selection of suitable technologies supporting the former for a given use case?

The first research question $\mathbf{RQ_1}$ targets the construction of a customized BI architecture in general, which can answer given business questions and their requirements. For that, an appropriate template forming a suitable basis for this construction process is required — a universal BI reference

architecture. Neither for Big Data nor for BI does a universally agreed upon BI reference architecture exist. Such an architecture should subsume both *novel* and *traditional* technologies. For this reason, the notion of *taking a traditional BI background into account* aims to explicitly include the knowledge, technologies, and approaches which have emerged for traditional BI such as RDBMSs and Data Warehouses. Notably, such a universal BI reference architecture needs to be built so that the connection to the process of developing said customized BI architecture can be explicated.

$\mathbf{RQ_2}$ focuses on this particular process of selecting appropriate technologies fitting the properties of a use case. This is to address the fact that the abundance of tools and their possible uses require additional information to be brought into the construction of a customized BI architecture, which relies on the selection of one or more technologies to support it. This involves examining the properties of traditional and novel technologies, which become part of a universal BI reference architecture, analyzing under which circumstances they should be used. These can then be mapped onto properties of a given use case.

These questions are answered by means of a newly constructed research artifact, the *GOBIA Method*.

1.2 Structure and Research Method

In this section, the structure of this work is described. This is followed by an introduction of the used research method, design science (cf. [PTRC07]), which is explained and related to the structure of this work.

After this, the *Background* chapter Chapter 2 introduces technical and organizational foundations for the upcoming parts of the work. Especially the technical foundations include a detailed view on both traditional Data Warehouses and novel Big Data technology to establish an overview of the various technical opportunities. Included here are also descriptions of basic concepts, such as relational databases and a declarative query language for them. Simultaneously, important phenomena and concepts around these technologies and building upon them are explained, such as Big Data in

general or the specifics of BI and its disambiguation to other related terms. Also in the *Background* chapter, the architecture analysis perspectives, such as technical, economical, or organizational ones, are derived and outlined, as they are used in the main part afterwards. Additionally, process modeling and requirements engineering are introduced as prerequisites for the later work.

This is followed by the *Analytical Architectures in Research and Practice* chapter, the first main chapter of this work. Here, an overview of literature and related work pictures reference architectures and architectures used for analyses by both academia and practice. This further motivates the need for a unified view on a BI reference architecture and points out commonalities among existing approaches. Moreover, using this and the technical properties established in the *Background* chapter, selection criteria for technologies are outlined. These criteria are used later to define a selection process for BI technologies. Lastly, all necessary requirements to the following proposal are laid out, using both literature on reference architecture formulation and all work made until then.

To explore the outlined research questions adequately, an appropriate research method must be chosen. In the Information Systems (IS) discipline, the two major paradigms of research methods are behavioral science and design science [HMPR04, p. 75] [MS95, p. 253]. Here, behavioral science aims to develop theories regarding past or future behavior of humans or organizations in the context of an information system, which are then verified. This is similar to research in the natural sciences, which also relies on building and testing theories [MS95, p. 253]. In contrast to this, design science is problem-oriented and takes an engineering approach to research while building upon existing theories attained through behavioral research. Its essence is to create "innovative" *artifacts* to "extend the boundaries of human and organizational capabilities" [HMPR04, p. 75] regarding "efficient and effective" [HMPR04, p. 76] development, usage, or management of information systems. Such artifacts can range from theoretical constructs such as ideas or best practices, to actually implemented information systems, as long as they are suitable for solving the problem of interest. March and Smith

summarize that design science is concerned with utility, whereas behavioral and natural science research focuses on theory [MS95, p. 255]. They, as well as Hevner et al., support the notion that newly developed artifacts can in turn be used to formulate and build new theories through behavioral science, thus conceptually intertwining both research paradigms [HMPR04, p. 76] [MS95, p. 255]. Also, Österle et al. endorse such design-oriented research in the Information Systems domain in general with regards to scientific rigor and effects on Information Systems research [ÖBF+11].

Therefore, design science presents the best methodological fit for this work. As the research questions aim for artifacts whose form is yet unknown and to be determined, the creation of such artifacts through design science research is, conclusively, an adequate approach to answer those questions and to solve the underlying problems.

Regarding the artifacts themselves, March and Smith detail four types of artifacts as outcomes of design science research in Information Systems [MS95, pp. 256ff.]:

1. **Constructs** form the vocabulary of a particular domain used to describe problems within it as well as define their solutions. Through conceptualization, this vocabulary entails a domain-specific language and shared knowledge in respective domain. Constructs can be specified both informally, e. g. through common behavioral patterns when working together in a team, and formally, where even highly formalized depictions are possible, e. g., a meta-model of a modeling language, which defines the constructs that can be used in the models themselves. This allows domain experts to express problems and solutions in a common way, facilitating the exchange of information and knowledge.

2. **Models** are abstracted representations of a real or an imaginary exemplar, the *original* [Sta73]. These abstractions, which leave out unnecessary aspects (i. e., they constitute a reduction), are not arbitrary, but are tailored to the goals of a model [MS95, p. 256] [SVOK12, p. 22]. Similarly, Frank et al. see models as "purposefully constructed [abstraction]" [FSF+14, p. 1] with the goal to reduce complexity and to

allow inspection and improved communication among stakeholders [FSF+14, p. 1]. As the use of models permeates everyday life and business, these goals exhibit an extensive variety, and can range from depicting a system in a specific manner (e. g., an architecture) to communicating between developers and other stakeholders in a project (cf. [SVOK12, p. 22]). Models used in engineering and construction as well as in natural science are often actual physical objects (e. g., a model car in 1:18 format or a smaller scale model of a building) [FSF+14, p. 1]. In contrast to that, models in the Information Systems domain are "linguistic constructions" [FSF+14, p. 1], which use constructs to express relationships between them. Conclusively, models can not only be represented visually (e. g., as in a typical process or data model), but also textually [MS95, p. 256]. Similarly, BI architectures are often visualized by models as well (cf. Section 3.1).

3. **Methods** are sets of steps to execute a given task [MS95, p. 257]. These steps can be algorithmic or take the form of a guideline. Methods are grounded in the underlying constructs of their domain and use models as representation of both task and result. Inputs into a method can also be models [MS95, p. 257]. For instance, the process of transforming requirements into increasingly technical forms until they eventually become program code can be specified using a method, which describes the involved steps of this transformation. Naturally, the choice of method influences both constructs and models created as result of working on a task, because methods imply a certain perspective while finding a solution [MS95, p. 257], e. g., a technical perspective or a business perspective to a development project.

4. **Instantiations** realize an artifact in the form of an information system. Simply put, they represent an actual implementation in the target environment of the underlying constructs, models, and methods. As March and Smith point out, instantiation does not necessitate deliberate explication with respect to the aforementioned artifacts, i. e., the underlying artifacts do not need to be represented distinctly. In-

> formation system can be implemented with these artifacts being tacit knowledge of the implementers. In this case, constructs, models, and methods need to be "reverse-engineered" theoretically or empirically if they should be formalized [MS95, p. 258].

Concerning the targeted contributions in this work, two artifacts are in the primary focus, namely models and methods. RQ_1 aims at a universal BI reference architecture, which needs to support the construction of a customized BI architecture. As architectures are usually visualized in a structured manner, a model is an appropriate artifact for a contribution to this question. RQ_2 targets a process to enable technology choices based on use case properties. This is a precise fit with the definition of a *method*. Hence, a method is also a relevant contribution candidate. Apart from that, constructs are in so far important as they are at least a tacit if not an explicitly formulated foundation for the development of models and methods. As various practical and research areas are potentially touched (e. g., traditional and novel sides of technology or business-related fields such as requirements engineering), established vocabularies should be employed so that these consistently and appropriately cover the contributed artifacts and their domains. Instantiations in the form of specific implementations and use cases of relevant BI technologies are eminently crucial to comprehend the use of certain technologies in their specific context and derive new findings, which are relevant for the aforementioned two main contribution artifacts. However, a full instantiation in the form of a realized architecture realized through hard- and software is not part of the present contribution itself as they are not in immediate scope of the research questions, although instantiations are discussed in several aspects as part of the development of BI architectures.

The specific research approach employed in this work adheres to the Design Science Research Methodology (DSRM) proposed by Peffers et al. [PTRC07]. Their methodology is well-known and design science in general is, as outlined above, also widely accepted for research in the IS area. Here, generally, an *artifact* is designed to solve a specific problem, which is outlined at the beginning. This artifact is then evaluated and shown to the research

community to iteratively improve it. The particular steps defined by Peffers et al. (cf. [PTRC07, pp. 52-56], see also Figure 1.1) are:

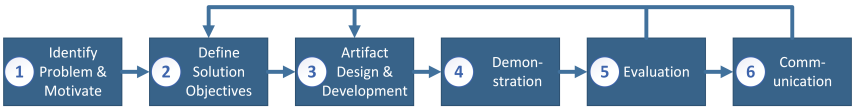

Figure 1.1: Design Science Research Methodology (DSRM) process model, based on [PTRC07, p. 54].

1. **Identify Problem & Motivate**. The first step is to determine the research problem, or research gap, that needs to be solved with design science. Beyond just outlining the issue, it needs to be motivated to underline why it is worthwhile to design a solution for it and which contribution a solution can offer with regard to the problem. This ensures, that the problem is important and relevant, and that the solution can advance the research field. If required, the problem can also be decomposed into various sub-problems or sub-goals, which the designed artifacts can specifically address. [PTRC07, pp. 52,55]

2. **Define Solution Objectives**. Once the research problem has been defined, the focus shifts to the artifact, which will be designed to solve the former. After the previous step, it is already established that an artifact *can* address the problem. However, it must now be specified, *how exactly* the artifacts can contribute to it. This is especially important if there are other artifacts that attempt to solve the problem. Because of this, more detailed objectives for the solution artifact are to be defined here. To that extent, qualitative and quantitative goals are defined, which allow to comprehend resp. measure the contribution of the artifact. As the design science process is iterative, objectives can be adjusted after the evaluation step.

3. **Artifact Design & Development**. The main step is concerned with actually creating the contributing artifact. Peffers et al. perceive such artifact quite broadly as anything that is designed with the "re-

search contribution is embedded in the design" [PTRC07, p. 55]. This especially includes methods, models, concrete implementations, but also theoretical constructs. In addition to creating the artifact, its desired properties as defined in the solution objectives have to be specified and the artifacts structure and inherent relationships are defined, i. e., its architecture. [PTRC07, p. 55]

4. **Demonstration**. The demonstration of the designed artifacts works as a "proof-of-concept". It shows whether the solution is feasible and fulfills the previously defined objectives and requirements to it. A demonstration could be a sample application of the designed artifact to one of the problems it was intended to solve, e. g., in form of a case study. [PTRC07, p. 55]

5. **Evaluation**. An evaluation is a more formal approach to test the appropriateness of the artifact. It especially includes a critical disambiguation of the desired objectives and actually reached objectives during a demonstration. It goes beyond a mere demonstration of an artifact by systematically recording and measuring its outcomes and comparing them throughly to expected values. The actual evaluation method depends on the research topic and type of work. Typical methods include empirical evaluations with quantifiable quality criteria for the solution or logical proofs, which check if the outcomes of the solution can be rationally considered to fulfill the objectives. If the results of the evaluation are unsatisfactory, the researchers should go back to the objective definition step and re-iterate the artifact to further improve it. [PTRC07, p. 56]

6. **Communication**. Lastly, the artifact should be communicated to researchers and other relevant parties, e. g., practitioners via respective outlets. This could be scientific- or practitioner-oriented publications. With feedback as result of the communication with these third-parties, further refinements and improvements could be triggered in the objective definition step or in the design and development step. [PTRC07, p. 56]

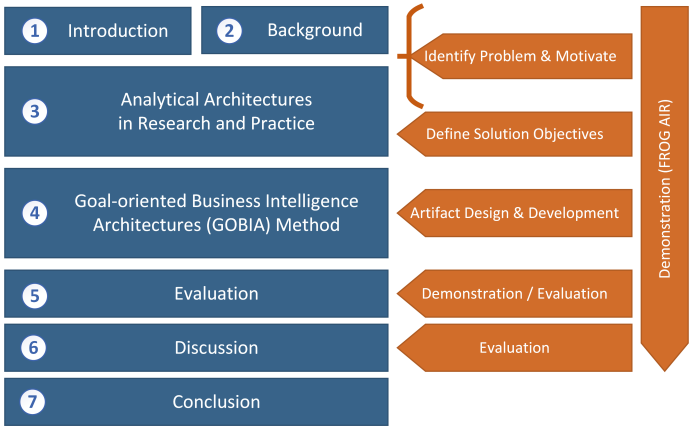

Figure 1.2: Steps from DSRM process model, based on [PTRC07, p. 54], mapped to the structure of this work.

This process is usually sequential, starting from the first step. However, Peffers et al. further note that later entry points can be chosen depending on the problem at hand. For instance, if a problem does not need additional motivation or is already well-outlined, one can already start with the second step and define solution objectives. If the research is based on already existing artifacts (e. g. to iteratively improve or adopt it for a different domain), the process can be started with the third step [PTRC07, p. 56].

Naturally, this description of the design science process is quite abstract. For this reason, the structure of this work is now mapped to the steps of the process (see Figure 1.2). As part of this mapping, it is illustrated if and how a step is specifically executed in the subsequent chapters.

This section as the introduction for this work is mapped to the step "Identify Problem and Motivate". Section 1.1 motivated the underlying issue, which is the lack of a unified BI reference architecture and an appropriate process to customize it. This motivation is underlined by examples from research and practice. The section briefly outlines the research gap and the goal of this work, which is to develop and propose a solution artifact addressing the

research questions above. However, the identification of the problem and the motivation for the proposed solution do not stop at this point. The subsequent chapter *Background* furthers the understanding of concepts that are used throughout the work, especially with respect to various technologies, which participate in a BI architecture. This facilitates an understanding of the issues that were outlined before. For instance, in case of BI technologies, technical details can be elaborated which contribute to the overall situation. This is continued in Chapter 3 on *Analytical Architectures in Research and Practice*. The analysis of the present state of BI architectures adds further arguments to the outlined research gap.

Additionally, the step "Define Solution Objectives" is addressed. After knowing the current state of the problem, requirements for the solution artifact — the GOBIA method — are specified. It is argued that a universal BI reference architecture alone is not sufficient; beyond that, a means to navigate through the current solution space is required. In the context of a specific use case, a set of feasible technologies from the BI reference architecture is chosen for a customized BI architecture with the help of a development process.

Chapter 4 features the design and development of the GOBIA method. It employs the accumulated knowledge to construct the artifacts, which comprise the method. This knowledge includes characteristics of traditional and novel technologies, existing forms of reference architectures and specific architecture. It is embedded into the design of the method as demanded the DSRM process. A running example (cf. Section 1.3) is employed throughout the chapter to demonstrate the use and utility of the artifact.

In Chapter 5, an initial evaluation of the GOBIA method is conducted using three case applications, which further illustrate the utility of the GOBIA method in a broader range of scenarios. Moreover, the solution objectives are reflected upon. The evaluation of the DSRM process concludes in Chapter 6, where the whole approach to this work is discussed and reflected upon. The final step of the design science research method, communication, manifests itself in the form of this thesis.

Finally, Chapter 7 concludes the work by summarizing it, highlighting and positioning its contributions, and outlining future research directions.

1.3 FROG AIR Sample Case

To illustrate the contributions of this work continuously, one use case is employed as a running example. This is in addition to the three separate case applications in the evaluation part. The running example is applied to each part of contributed method of this work in Section 4 and serves as means to demonstrate its efficacy as mandated by the *demonstration* step in the DSRM process.

The use case of the running example involves a BI system that is developed due to an adapted strategy of the involved company *MIDAS*. While naturally not being the sole artifact that aids in the fulfillment of the newly formulated strategic goals, the particular BI system is aligned to the updated strategic direction. The BI system itself is developed as a prototype independently of this work. However, this might allow the result of the method application to be compared to the actually implemented system. As the case and its planning process is based on a real project, names of all involved companies and other stakeholders are replaced by pseudonyms. Here, the specific system has a focused scope in terms of complexity and overall size. This focus should make the example more approachable and comprehensible later on, as the created artifacts can be comparatively less complex as well (i. e., goals, requirements, or model elements later on). In this section, a brief background on the case is given, highlighting the rationale of MIDAS as the involved organization behind the to-be-designed BI system.

MIDAS follows a new omni-channel Customer Relationship Management (CRM) strategy regarding its customers, for whom MIDAS manages the customer interaction. For instance, MIDAS is tasked with the social-, email- and phone-channel support for **FROG AIR**, an airline company. MIDAS uses its own systems for this, but also needs to use ones provided by FROG AIR (e. g., for flight booking data), which is outside MIDAS's control sphere. In general, an omni-channel strategy denotes the strategic approach to use a

variety of communication channels such as e-mails, social media, mobile, and online applications to contacts customers and interact with them (cf. [VKI15]). As natural consequence, a provider (i. e., MIDAS) collects data on customers in a variety of formats, be it structured or semi- or unstructured data. The omni-channel strategy aims to utilize this data for a strategic intent.

MIDAS's strategic goal in that sense is to offer a) added value to their customers through an omni-channel approach which leads to b) a competitive advantage over its competitors. As a side-condition, MIDAS would like to avoid costly custom solutions in the future and try to standardize its software portfolios regarding channel supports.

Due to this background, it was decided to develop a prototype BI system whose *overall goal* is to enable involved stakeholders to monitor customer satisfaction metrics of various social and other electronic channels that MIDAS provides for FROG AIR. The system is further referred to as FROG AIR Customer Service Monitor (CSM). Naturally, such a system could be expanded for other clients as well in the future.

The specific goals and requirements will be depicted after elaborating upon *requirements engineering* in Section 2.5.4. The demonstration using FROG AIR is conducted throughout Section 4.3.

2 Background

The following background chapter introduces important fundamentals for a better understanding of the subsequents parts of this work. First, the traditional form of Business Intelligence is elaborated upon in Section 2.1, where a common understanding of the term BI itself is established. Data Warehouses, most often associated with traditional BI, and Relational Database Management Systems and SQL as its technological foundations are explained in detail. The basics of Big Data technology and analytics are discussed in Section 2.2 and Section 2.3. The plethora of novel Big Data tools and technologies necessitates an overview of relevant exemplars in this area and a precise analysis of Big Data as phenomenon. Afterwards, analytics conducted in the context of Big Data but also with traditional technologies are elucidated. Following this, an overview of the topic of process modeling is given in Section 2.4, which is the fundamental for designing the proposed method. Next, the topic of requirements engineering is elaborated upon, as understanding the concepts of use case goals and requirements is integral to construction of this work's contribution. Lastly, the analysis perspectives used throughout this work are presented in Section 2.6. These perspectives are used to examine aspects of BI reference architectures.

2.1 Traditional Business Intelligence

Business Intelligence has become a ubiquitous term that heavily builds on IT support. However, first mentions of BI can be traced back to at least the 19th century. In 1839, O'Brien used the term in a title of his book [O'B39] about "Wholesale Business Intelligence". Later, in 1865, BI was employed to describe an example of banker Sir Henry Furnese, who efficiently gathered business-

relevant information and acts upon it for competitive advantage [Dev65, p. 210]. As standalone word, "Intelligence" has an extensive history as well. Besides denoting the cognitive potential of a human mind, it commonly refers to the strategic acquisition of information of military or political value [320]. Naturally, this has been conducted since a much earlier time than the 19th century and it still done in present times, albeit through dedicated organizations (cf., e. g., [Fin98]). For instance, government agencies such as the Central *Intelligence* Agency (CIA) in the United States [130] fulfill these intelligence-related tasks.

Defining Business Intelligence

Before it can be disseminated what constitutes "Traditional" BI, light has to be shed upon the term BI in general. While there exists a shared notion of what BI is basically about, several definitions were proposed, which differ in various points. Some of these are briefly discussed in the following. In general, several understandings of BI relate the term more or less closely to the underlying technologies and systems such as Data Warehouses. Nevertheless, BI can also be viewed as integrated approach throughout an organization, which involves aspects such as strategies, processes, or organizational elements that go beyond the mere technology of a Data Warehouse.

Interestingly, even BI as it known and understood today, was already coined in 1958 by Luhn, years before the start of widespread Computerization and IT usage in organizations. In his definition, key-elements of traditional BI technology are described, such as a BI system being "an automatic system" to "disseminate information to various sections of any [...] organization" [Luh58]. The underlying notion of the defined "intelligence system" is that intelligence would improve business understanding so that "action towards a desired goal" could be guided [Luh58, p. 314].

More recently, several sources (e. g., [GRM15, 334, 250]) attribute a definition of BI, from allegedly 1989, to Howard Dresner. In spite of this, including a widespread agreement regarding it and Dresners influence on coining the term BI [Pow04, 335], an original source of this definition is never mentioned explicitly, not even by Dresner himself [162], and thus

it cannot be verified conclusively (cf. [275]). This definition phrases BI as an umbrella term for "a set of concepts and methods to improve business decision making" [GRM15, p. 1; paraphrasing, supposedly, Dresner]. Similarly, Gartner Group, defines BI as "an umbrella term that includes the applications, infrastructure and tools, and best practices that enable access to and analysis of information to improve and optimize decisions and performance." [180]. [GRM15] summarizes that through various definitions, the intentions of BI remain the same. According to them, BI summarizes a "set of models and analytical methods" [GRM15, pp. vii].

The main goal of BI is to provide "decision support for specific goals defined in the context of business activities in different domain areas taking into account" based on "empirical information". Notably, "business" is to be understand broadly across domain boarders, be it scientific projects or commercially-oriented use cases in enterprises [GRM15, pp. vii, 1f., 4]. It ranges from operational support (e. g., in a specific machine) to strategic top-level support to decision makers (e. g., using sales Key Performance Indicators [KPIs]). Naturally, this empirical information has to be gathered and prepared by means of IT artifacts, before being analyzed [GRM15, p. 2]. Although BI is often associated with certain analytic techniques [GRM15, p. vii], it goes beyond them. Newer definitions, but also the previously mentioned ones by Gartner Group and Dresner, convey that BI is a holistic IS approach to decision support. This means that it covers the company as whole, including organizational and process-related aspects (cf. [BK08, p. 132] [KBM10, pp. 8f.]). For this, it is necessary that any to-be-designed BI system adheres to the business goals applicable in the context of the system (cf. [GRM15, p. 5]).

Therefore, for this work, BI is defined as an integrated, organization-specific, holistic IS approach for business decision support[1]. Importantly, when BI is rooted as an Information Systems approach, its integrative role is strengthened. This is because an IS is a socio-technical system, which

[1] This definition is based partially on [KBM10, p. 9] (translated from German), notably replacing IT with the broader IS, as well as [BK08, p. 132] and on the discussion in [GRM15, pp. 1-3].

encompasses not only IT artifacts as technical "executors" for BI, but also human and organizational elements (functions, structures, and processes) (cf., e. g., [ÖBF+11, p. 8]).

With this definition in mind, it is clear that a specific BI system or tool is not equal to BI as whole. A *BI system* is only part of a holistic BI approach. A Data Warehouse is one typical instance of a BI system. Certain BI tools and methods can thus be used to develop BI applications and systems. A DWH reference architecture is one such tool. (cf. [KBM10, p. 9])

This notion of BI is also not focused on a specific technology. Thus, using such a "broader" definition, both traditional and new technologies (cf. Section 2.2) as well as well-known and novel organizational processes (e. g., cf. Section 3.2) can be part of a BI system. This underlines the desired understanding that BI is technology-agnostic in nature.

Role Development of Business Intelligence

BI was initially associated with so-called Decision Support Systems (DSSs). DSSs were the predecessors of modern analytical systems and were incepted in the 1960s [GRM15, p. 2]. What followed starting in the 1990s, and is classified today as *Traditional BI*, was "data-driven" decision support that relied on technological artifacts and analytical concepts, which are described in the following sections: DWH, OLAP, and Data Mining. Grossmann and Rinderle-Ma note that especially in this era, a narrower understanding of BI was formed. It focused especially on technical capabilities and associated business tasks of the aforementioned analytical tools, which would have equated a rather heterogeneous tool landscape with BI. This narrow definition is also employed when BI is seen synonymously to the underlying technologies, such as a DWH.

For instance, in [KR13], Kimball and Ross use Data Warehousing and BI conjointly ("DW/BI"). Although they see the scope of this beyond technology and tools (e. g., with regard to business needs), their notion of a BI system resembles essentially a DWH (cf. [KR13, pp. 1-4]). "Traditional BI" is associated with such narrower, technology-focused understanding of BI. While this work is based on a broader BI definition as stated in the previous

paragraph, an examination of this traditional understanding through its key technologies helps to better comprehend both recent developments in the BI and Big Data fields as well as the contribution of this work (cf. Chapter 4).

The following two sections describe relational databases as key foundation technology in Section 2.1.1 and Data Warehouses, in conjunction with OLAP and Data Mining, in Section 2.1.2 as cornerstone of Traditional BI.

In subsequent sections, this traditional notion is amended technically with aspects of a "modern" BI, especially regarding enhanced analytics capabilities and techniques (cf. Section 2.3). Moreover, on an organizational level, "agile BI" also tackles the planning side of BI with aspects from agile software development (cf. Section 2.5.3).

2.1.1 Relational Database Management Systems and SQL

A Relational Database Management System is a prominent representative of a "general-purpose software system" [EN16, p. 6] known as Database Management System (DBMS). A DBMS acts as an interface between end-users and one or more databases [Vos91, p. 14] [EN16, p. 6]. End-users can be actual human users or applications accessing the databases via a DBMS [Vos91, p. 7]. A database is a collection of formatted data [Vos91, p. 4] (cf. [EN16, p. 4]). Together, DBMS and databases form a Database System (DBS) [EN16, p. 6] (cf. Figure 2.1). DBSs are a "widely accepted tool for the computer-aided management of large, formatted collections of data" [Vos91, p. 4]. Relational DBMS and databases are based on the relational data model, which describes the logical organization of data in the database.

Background and Architecture

RDBMS reached the mass-market in the 1980s and had considerable success since, receiving widespread attention and use from both academia and practice. Historically, they are considered the *fourth generation* of database systems [Vos91, p. 7]. In this generation, the term DBS was finally manifested, with DBMSs becoming the key software application mediating between users and data. The separation between physical and logical views

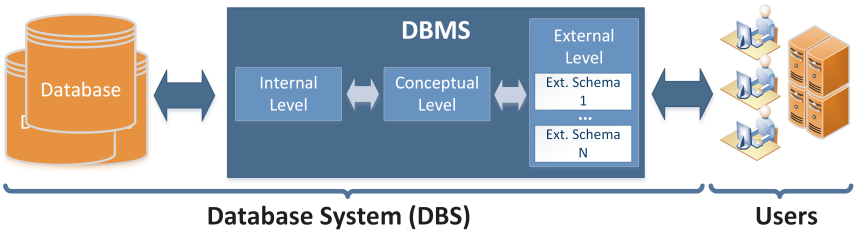

Figure 2.1: DBS usage scenario with a three-level architecture DBMS, databases, and users, both humans and machines. Based on [Vos91, pp. 7;12] and [EN16, p. 37].

became common with this generation. Especially, the so-called *relational data model* fueled this clear separation. With the wider spread of RDBMS, data models started to receive more attention. Today, they are both a topic of ongoing interest and activity from both research and practice. For instance, conceptual modeling languages such as entity-relationship models are often used to model data (visually). Data models further enforced the separation between physical values and syntax and semantics of its description (i. e., the "(logical) meaning of data") [Vos91, p. 7]. This also brought storage transparency, meaning that the concrete form of physical storage could theoretically change over time, without the applications using the logical model being affected by this.

This distinction also allowed an integration of all relevant application data into one collection. As per definition from above, this constitutes a database. These properties are connected to the goal of *data independence*, which is desired when introducing a DBMS (cf., [Vos91, pp. 10f.]).

These separations resulted in the so-called *three-level architecture* [Vos91, p. 12], which was originally proposed in [ANS75]. It features three strictly separated layers[2], which are called levels and fulfill different roles in data organization. The three levels are schematically illustrated in the overall DBS context in Figure 2.1. The lowest level is the *internal level*, where physical

[2] Architectures in general and layered architectures are further described in Section 3.1.

data structure and organization is aligned to the underlying hardware [Vos91, p. 13] [EN16, p. 37]. The *conceptual level* contains the conceptual schema ("logical general view"), which founded in a *data model* such as the relational data model. Using this data model, it forms a logical abstraction to physical access. Access languages such as SQL are used to query data [Vos91, p. 13] [EN16, p. 37]. The *external level* may represent an external view as a subset of the full conceptual schema. This makes the underlying overall data model is transparent, rendering it interchangeable [Vos91, pp. 7f.; p. 13]. However, a separate external schema is not always needed and the difference between this and the conceptual layer remains theoretical [Vos91, p. 13].

Relational Data Model

The relational data model used in RDBMS is based on the work of Codd [Cod70].

In a relational DBS, a database consists of *tables*. In a table, data is contained in form of *rows*. A row denotes a set of related data. For each table, there is some form of "table header", which denotes the *attributes* in the *columns* of the tables. Attributes are regarded as time-invariant. All rows except this header line contain the attribute *values* (or attribute instances) for each of the attributes defined in the header. The values are, in contrast to the aforementioned header, time-variant. (cf. [EN16, p. 179])

The data in the example table *RepairJobs* depicted in Table 2.1 can be readily mapped to these terms. The header consists of the four column names "RepairID", "Car", "Customer", and "Cost". There are five rows with respective values. For instance, the first row contains "BMW 318 Ci" as value for the attribute "Car" and "2000" for "Cost".

Codd uses *relations* from mathematics (more specifically, set theory [EN16, p. 153]) to create a relational view on data. Here, a database is represented by a collection of relations. A mathematical *relation* is denoted by $r(R)$ and is a set of *n-ary tuples* without particular order, while r is the relation itself and R is the *relation schema*, which r adheres to. The schema R is defined as $R(\mathbf{A}) = A_1, ..., A_n$, where as $\mathbf{A}$ is a set of attributes. Each attribute A_i has a *domain*, $dom(A_i)$, specifying the range of possible values for the attribute.

Table 2.1: Relational example table *RepairJobs*.

RepairID	Car	Customer	Cost
1	BMW 318 Ci	David Black	2000
2	Trabant 601	Gerald Vannen	150
3	Audi 80	Peter Pan	375
4	BMW 318 Ci	David Black	500
5	Mercedes E190 2.0	Peter Kontes	200

For instance, the domain of an unsigned integer attribute A_{uint} may be defined as $dom(A_{uint}) = \mathbb{N}_0^+$ (i. e., the set of positive natural numbers including zero). Essentially, a domain is represented in an RDBMS primarily through a *(data) type* definition (among with other data constraints) [EN16, p. 151]. With this information, the relation r can be defined as a subset of the Cartesian product of all its attributes' domains: $r \subseteq dom(A_1) \times ... \times dom(A_n)$. In simple terms, such relation r is made up of combinations of possible values for each of its n attributes, the latter of which are given by their domain $dom(A_i)$.

An *n-ary tuple* in a relation has exactly n attribute values. It is an *ordered* set of these values. Each attribute value a_i is an element of the attribute domain $dom(A_i)$ of attribute A_i. All attributes that are part of a tuple can be defined by the set $A = \{A_1, ..., A_n\}$. More formally, a tuple is the injective and total mapping $\mu : A \rightarrow dom(A)$ for which the following holds: $(\forall S \in A)\mu(S) \in dom(S)$ [Vos91, pp. 98f.]. It should be noted that dealing with empty cells, known as *null values*, requires a slight adaptation of these definitions (cf., [Vos91, pp. 99f.]).

The concepts of (primary) *keys* and functional dependencies (FDs) as well as *normalization* are crucial in relational models. *Primary keys* refer to attributes which identify or characterize a tuple uniquely. Keys and FDs help to organize relations and minimize redundant data with so-called *normal forms* [Vos91, p. 216]. For a more in-depth perspective on these aspects, respective literature can be consulted, such as [Vos91] or [EN16].

Operations, such as querying, on relations are formulated using *relational algebra* [Vos91, p. 114]. They are based on mathematical set operators. These can be used, among others, to select a subset of one relation or of multiple relations combined. Essentially, they define a requested sub-portion of the data. Besides combining the values of several relations, data not necessary to answer a query can be left out. For example. a *projection* π on a relation returns a subset of the relation attributes (e. g., $\pi_{Car,Cost}$). A *selection* σ is to apply filter conditions to return a subset of the relations' tuples (e. g., $\sigma_{Cost>500}$). With a *join* operation, multiple relations can be combined into one greater (joined) relation. To be able to perform such join, a common domain must be present under which the join can be performed. A comprehensive list of operators and further aspects to relational algebra and calculus can be found not only in Codds original work ([Cod70]), but also in several other literature sources (e. g., [Vos91] [EN16]).

Apart from that, on a higher level of abstraction, conceptual modeling can be used to design data models and leave out some technical details initially. Here UML diagrams (cf. [EN16, p. 60]) or entity-relationship models (ER models) can be used [Che76]. More recent *enhanced ER (EER)* models [EN16, p. 107] work with entities such as customers or products and relationships between those (e. g., in form of a sales order). The EER models can be converted ("mapped") into a relational model, where entities and relationships are explicated as tables and foreign keys. Tools for such activities include, e. g., MySQL Workbench [314]). [Vos91, pp. 35ff.] [EN16, pp. 289f.]

SQL

Structured Query Language (SQL) is a query language used to query data from a relational database. Instead of offering direct programmatic access to relations, tuples, and values directly via an Application Programming Interface (API), relational databases allow to send them queries written in SQL as text. These queries are interpreted and executed in the RDBMS. SQL builds on the principles of the relational model to express the desired result in relation to the exposed external or conceptual schema.

SQL is a declarative language. Unlike in imperative programming languages such as Java or C++, users of a declarative language describe *what* they expect as result from their command (e. g., a selection on a Cartesian product of two tables as above). An RDBMS executing a SQL query chooses an appropriate way to execute the query physically and returns the result. RDBMS query executors employ cost-based query optimizers [SC05, p. 3] [EN16, pp. 43f.], which take into account several statistics about the underlying databases accessible through meta data, and choose one of several possible *query plans*, which describe the physical execution of a query. The generation of candidate plans takes place under both time (i. e., time in which the plans are generated) and resource constraints (i. e., resource costs of executing the plan) [Nev11, pp. 17f.]. For instance, it would not be desirable if query optimization took one minute to save ten seconds of query runtime.

In general, SQL allows querying (i. e. reading via **SELECT**) data as well as changing it (i. e., create new rows via **INSERT**, modify existing ones via **UPDATE**, or delete them via **DELETE**; cf. [EN16, pp. 198ff.]). Hence, SQL is classified as Data Manipulation Language (DML). In addition to this, SQL allows to define database schemas, which include definitions for databases (**CREATE SCHEMA**), tables and their columns (e. g., **CREATE TABLE**), views (e. g., **CREATE VIEW**) and other database objects [EN16, p. 54]. Hence, SQL is also a Data Definition Language (DDL) [EN16, p. 178].

SQL has been standardized in the International Organization for Standardization (ISO) \ International Electrotechnical Commission (IEC) 9075 standard since 1987 [ISO87] with SQL:2016 being the latest version [ISO16]. However, not all vendors strictly adhere to the standard (e. g., by differing how NULL or dates values are handled). This leads to several SQL "dialects", which can limit transferability of SQL statements. For example, the RDBMS MySQL offers an extensive list of alterations from the official standard (e. g., [304] for version 5.7). Notably, they do not commit to any specific SQL standard conformance. Only earlier documentations claimed "entry-level

```
SELECT
  Car, SUM(Cost)
FROM
  RepairJobs
WHERE
  Customer = 'David
  ↪  Black'
GROUP BY
  Car
```

Car	SUM(Cost)
BMW 318 Ci	2500

Listing 1: SQL aggregation with selection example using GROUP BY and WHERE with query on the left and example table *RepairJobs* output on the right.

SQL92" support[3] while striving to support SQL:99 fully [WAM02, p. 36]. In contrast, PostgreSQL 10, another RDBMS, claims to almost completely support SQL:2011 [330].

A demonstration of an SQL query using the relational table in Table 2.1 is shown in Listing 1. It uses an SQL selection[4] and a **WHERE**, where conditions to fulfill are specified. These conditions usually refer to columns of tables in the **FROM** clause. Furthermore, an aggregation on several using a **GROUP BY** clause is performed to sum of all repair costs by car. Due to the **WHERE** condition, only results for customer "David Black" are included.

Besides **SUM**, a set of other descriptive operators are available such as ones for average (**AVG**), counting rows (**COUNT**), variance (VAR), or standard deviation (STD) (e. g., cf., [300, 265]). In addition to that, modern RDBMS and SQL standards allow for other sophisticated features and operations, such as nested queries, views ("virtual tables"), triggers, and assertions (cf. [EN16, pp. 207ff.]).

[3] SQL:92 only defines "entry", "intermediate", and "full" sets of conformance to the standard. Later standards have several individual features which can be supported separately until full conformance is reached [ISO92].

[4] SQL uses the term "selection" and the **SELECT** keyword for a relational projection.

Transactions and ACID Properties

Any SQL query is executed within a so-called *transaction*. In this context, a transaction forms a "logical unit of database processing" and is executed in a DBMS. Naturally, more than one DML operation may be necessary in a transaction. Transactions can either be created directly by the calling application or embedded into SQL or other query languages [EN16, p. 747]. When all SQL commands after a signaled start of a transaction[5] have been issued, a transaction can be applied by committing it. Unsuccessful transactions lead to a rollback, where any pending change is undone [EN16, p. 754]. In this sense, database transactions are the execution of programs, which represent all necessary steps for actual business transactions, which are simply "an interaction in the real world, [...] where something is exchanged" [BN09, p. 1], usually between natural persons and organizations (e. g., buying a product is an exchange of said product against money) [BN09, p. 1].

RDBMSs offer several guarantees with regards to the behavior of database transactions. These guarantees ensure that certain side-effects, for instance through concurrent access to a database, can be remedied. Applications that feature a high number of concurrent accesses, often caused by many users, include systems such as banking or finance (e. g., stock exchanges) systems, online retail systems, or travel booking systems (cf., [EN16, p. 745]). Such transaction processing systems need not only high availability, but also consistency.

To implement these safeguards, RDBMS transactions have properties known as *Atomicity, Consistency, Isolation, and Durability (ACID)*, which should be (and are) enforced by RDBMSs [EN16, pp. 757f.]:

1. **Atomicity**. This expresses that a transaction is either fully completed or not at all. It should be impartible like an atom. When there is an error midway during the execution of a transaction, all already applied changes need to be rolled back to their previous state and the other

[5] Explicit control of transactions is often optional in RDBMS. E. g., in "auto-commit" modes each SQL statement is implicitly executed within its own transaction and automatically committed after being issued (e. g., cf., [302]).

pending changes are canceled. For example, if bank transfer is aborted midway, money is removed from the sender, but not booked for the recipient. Such results are undesired. [EN16, pp. 757f.]

2. **Consistency**. Starting in a consistent state, a transaction should leave a database in such state after execution as well. Consistency especially refers to values having the correct domain (i. e., respecting the data type and other value constraints; schema constraints), but also to referential integrity constraints, especially correct primary and foreign keys. [EN16, pp. 757f.]

3. **Isolation**. This property means that transactions are isolated from one another, especially when concurrent transactions occur (e. g., by other users). The transaction should be executed as if the other transaction(s) were not there and vice versa, i. e., without interfering with each other. [EN16, pp. 757f.]

4. **Durability**. A transaction should be durable in a sense that changes by transactions are permanent. Even if any kind of fault occurs (e. g., a power outage or application crash) after a transaction is reported as completed, its changes are sustained and are not lost. [EN16, pp. 757f.]

The *ACID* guarantees in RDBMSs are a pillar of many business applications, because sensitive business processes deal with high-valued and mission-critical data, e. g., a bank transfer or an online sale (cf. [369]). Here, incomplete or contradicting business transactions should be avoided (cf., [BN09, pp. 1f.]).

Mechanisms in RDBMSs to implement transactions include approaches for locking parts of the database from concurrent access such as the *two-phase locking* protocol [Vos91, pp. 406f.]. Usually, RDBMS allow for certain control of transactions isolation levels (e. g., cf. [ISO92]), trading complete transaction safety for better performance by accepting side effects such as the possibility of reading uncommitted changes from other transactions ("dirty read") (cf. [Vos91, pp. 393ff.]). Moreover, recovery after failures (e. g., power outage or hardware failure) is important for RDBMS (e. g., cf. [EN16, pp. 813ff.]).

Usage Scenarios and Extensions

An RDBMS is a general-purpose system. It supports a wide range of applications. Especially due to the relational data model, many use cases can be covered. This is evident when considering the widespread use (e. g., cf. [6]) and research interest in RDBMS.

Cetintemel and Stonebaker summarized this claim of universal applicability of RDBMS in the preceding decades as "one size fits all" [SC05]. Notably, although RDBMS are still applicable to many use cases, the notion that RDBMS fit "all" has been challenged in recent years [SC05]. This can also be seen through the emergence of new technologies, which offer alternatives to the relational model, SQL, and strict ACID guarantees (see Section 2.2).

However, a more specialized and often employed purpose of RDBMS are OLTP applications. OLTP is characterized through its focus on transactions in particular, of which there are many concurrent ones (e. g., in a concert ticket booking system), which should be executed without additional delay, i. e., as fast as possible [EN16, p. 52]. This is particularly useful for business applications [SC05]. An *on-line* transaction underlines the necessity for an "immediateness" in processing it. For example, when booking a concert ticket, one can expect a timely answer to a booking request. In contrast to that, *off-line* transactions are usually employed in context of batch processing. Here, transactions are collected in "batches" and executed separately, leading to additional delays. The separate batch execution can still take place in a relatively quick succession (e. g., in ten seconds), but one might choose a different timing (e. g., one a day or even more seldom)[6]. Thus, an OLTP needs to be able to cope with high data volume efficiently [BN09, p. 2]. This is in addition to the needed consistency through transactions.

RDBMS and their predecessors are designed for storing and processing textual contents (e. g., actual text or numbers), while offering limited support for binary data in Binary Large Object (BLOB) columns. Some modern RDBMSs have added support to work with other types, such as Extensible Markup Language (XML), JavaScript Object Notation (JSON) (cf. Section 2.2.1), or

[6] The temporal aspects of batch processing are further considered in the technologies discussed in Section 2.1.2 and Section 2.2.4.

geo-spatial data [331]. An extended discussion of further advancements to RDBMS, tuning them for Big Data workloads, is conducted in Section 2.2.7.

2.1.2 Data Warehouses

The idea of DWHs is driven by the need to collect, integrate, and analyze data from operational systems in a central place. Historically, DWHs are separated from these systems, because they conduct long-running and complex queries and other related tasks. Operational systems, which are day-to-day productive systems, should not be negatively impacted in their performance because of this. Other reasons to keep the systems separate are that operational systems may each have their own, tailored data formats and that the underlying technology for operational processing may be different from analytical technology for systems such as a DWH (cf., [Inm05, p. XIX]). This section details the most important aspects and properties of a traditional DWH.

DWHs have an extensive, decade long history and are widely researched by academia and used in practice. Notable contributions to the field of DWHs are made by Inmon [Inm05] and Kimball [KR13]. In building a traditional DWH an "Inmon approach" and a "Kimball approach" are distinguished [Inm05, p. 357].

While several definitions of DWHs exist (e. g., [CD97, KR13]), a prominent one[7] is provided by Inmon. According to him, a DWH is a "subject-oriented, integrated, nonvolatile, and time-variant collection of data in support of management's decisions." [Inm05, p. 29].

This definition by Inmon is used for this work. It explicates several important characteristics that help to classify an analytical system as DWH. Furthermore, the definition is independent of technology. Other authors' depictions also share these traits, e. g., [CD97]. Chaudhuri and Dayal more broadly describe Data Warehousing as a "collection of decision support technologies" [CD97, p. 1]. Interestingly, such broader definitions bring it closer to the aforementioned narrow, technology-focused understanding of tra-

[7] For his whole book series, starting in 1992, Google Scholar counts more than 6000 citations [190].

ditional BI. That further underlines the necessity to properly disambiguate the term BI, as done prior through its holistic understanding. Kimball's definitions is also broader than Inmon's. It refers to a system specifically designed and structured for outputting data ("getting the data out", cf. [Kim96, p. xxv]), which was inputted into operational systems and adheres to goals such as consistency, capability for querying and analysis, and quality [KR13, pp. 3ff.].

Operational systems are often OLTP systems (cf., Section 2.1.1) with respective workloads. They are, e. g., used by many users, write small updates and read only a few records. Hence, queries are rather predictable and less complex [Kim96, pp. 1f.]. Often repeated banking transactions (e. g., Person A wants to transfer X € to Person B), sales transactions, and other business transactions, follow similar if not the same patterns. This influences planning of such systems, as they have to handle large workloads due to many users [BN09, pp. 17f.]. However, it is usually a finite set of transactions that is used. Users do not regularly issue custom ad-hoc queries to these systems. This set of transactions is regulated by underlying business processes that a respective application implements [BN09, pp. 121ff.]. In case of a retail sale these are typically pre-sales (e. g., browsing a product website), sales (e. g., buying a product), and post-sales processes (e. g., creating a support ticket). The persons executing such business transactions are often the ones directly involved in them, e. g., sales workers, clerks, customers, or IT personnel [BN09, pp. 1ff.]. OLTP transactions are measured by performance (high transaction throughput), availability, and consistency (cf. ACID in RDBMS) [Kim96, pp. 2f.].

In contrast to OLTP systems, DWHs are regularly used for OLAP. OLAP features an analytical workload, which is different from OLTP. It is characterized by few users conducting analyses, which consist of read operations affecting data from numerous transactions. Queries are long-running, complex and can also be formulated ad-hoc, which means they are freely specified by users and are not as static as in OLTP transactions. Consequently, the user group involved in OLAP is also different. These users are *knowledge workers* such as analysts, managers, and other executive staff [CD97, p 1] [EN16,

p. 1102]. They are mostly neither customers nor the company's employees involved in the analyzed transaction data. DWHs are also significantly larger in size than their source systems. Naturally, one reason is that they integrate data from all these sources. The integration of this data, updating and inserting new records, is separate from the read-focused queries issued by the users. Traditionally, this integration is done in regular batches (e. g., intra-day, daily or weekly). In addition to that, to provide a historically correct perspective on data, DWHs keep data for long periods of time, while it possible to delete unnecessary data in OLTP. Their analytical focus emphasizes the use of DWH as BI system for decision support (also referred to as DSS [EN16]). In the last decade of the previous millennium, a DWH of 50 to 100 GB was considered large, while nowadays DWHs can reach sizes on Petabyte scale. Hundreds of GBs or several TBs are hence not an untypical size [Inm05, p. XXV; p. 331] [Rus11, p. 16].

Characteristics

DWHs as OLAP systems are separated from actual operational data sources, which are fed into the DWH. Characteristics of a DWH, as stated in Inmon's definition, necessitate this separation. Prior to DWHs, corporate users would resort to individual *"extract" processing*, where they extracted required portions of data from one or several data sources ad-hoc. This already allowed a separation of the extracted data from their high-performance operational data sources. Thus, data could be further processed or analyzed without impacting the source system. However, such unstructured and uncoordinated individual extractions have several disadvantages. Through "natural" growth of database architectures at a firm with several applications and databases, Inmon postulates that such evolving overall architecture resembles a "spider web". Here, once extracted data becomes part of new databases, from which in turn data might be extracted for other uses as well. This would lead to issues with data credibility, productivity, and hinder the organization from transforming data into useful information. [Inm05, pp. 5-12] Early forms of DWHs can be traced back to 1983, but DWH in their usual form and as described here only became commonplace starting in the mid-1990s with

appearance of adequate data integration and analytics approaches for DWHs [Inm05, p. 402].

While a DWH maintains the advantage of extracting data so that the source system does not exhibit performance drawbacks, it helps to alleviate the aforementioned issues that stem from a lack of a central, structured organization for these individual extraction practices.

Subject-oriented data in a DWH means that the organization of data and entities is geared towards the entity, i. e., the "subject", in question, as defined in the data model for it. In contrast to this, operational sources have an application-tailored data model. For a subject *customer*, all subject-related data is usually bound by a common identifier (e. g., `CustomerID`), even if data is distributed to several tables or databases. [Inm05, pp. 33ff.]

A pure usage-based ("ad-hoc") extraction of data means that integration of several data sources has to be potentially repeated, when a different kind of extraction is conducted. Differences when combining several data sets can lead to inconsistencies. DWHs are *integrated*. This means that they harmonize all differences from the operational sources. Contradictions and inconsistencies in several source data sets are resolved so that data in a DWH can become the single point of truth in a organization.

The concept of *non-volatility* underlines the fact, that in systems with analytical workloads, as a DWH, data remains unchanged, while in OLTP systems, updates and deletions of data are common. Moreover, transactions often work on a record-level (i. e., they change one or only a few rows), whereas data is typically loaded in large batches into a DWH, where it remains and is not overwritten by subsequent loading of new data from its data sources [Inm05, pp. 31f.]. Importantly, end-users do not have access to this ingestion process.

While operational systems are employed for day-to-day operations and execute business processes, which are bound to change, such systems and their databases and especially the values are changed regularly or at least from time to time. Hence, OLTP systems work on shorter time frames, whereas a DWH is designed as *time-variant* for long-term usage (5 to 10 years is a common range [Inm05, p. 66]), such as for yearly figures or KPIs in comparison

over several years (cf., [Inm05, p. 36]). Inmon sees an operating time range of 60 to 90 days as "normal" [Inm05, p. 36], however archives inside operational systems can also hold data for up to a few years [Inm05, p. 66], depending on the use case (cf. [Inm05, p. 67]). For operational systems, the current values are the primary focus and old values may be plainly overwritten by new ones, while a DWH works on what are essentially snapshots of data loaded into it in order to reflect on changes and developments over time [Inm05, p. 36]. For instance, an organizational structure embodied in a source database, resides there only in its latest form, because older ones are not relevant for business. On the other hand, a DWH can store time-variant snapshots (cf., [Inm05, p. 36]) of these structures to see historically correct figures for older data, where necessary. In this example, in case of organizational restructuring, an operational system might not be able to show its figures in relation to the older structure, whereas a DWH with an explicit concept of time can use both old and new organizational structures to calculate the needed KPIs.

Extraction, Transformation, and Loading (ETL)

A DWH can be fed from several types of data sources, which can be internal sources from within an organization or external data sources such as stock market data supplied by a data provider (cf. [Inm05, pp. 258ff.])[8]. One common source are relational databases, which are used by OLTP applications. Other types of data sources include text file formats (e. g., spreadsheets), legacy systems (e. g., hierarchical databases, cf. Section 2.1.1). Depending on the needs of a DWH, appropriate sources are chosen and selected for extraction accordingly [KC04, p. 54].

To attain an integrated collection of data in a DWH, data integration is conducted with a so-called Extract-Transform-Load (ETL) process [Inm05, p. 18]. It is responsible for extracting data from data sources, transforming it to fit the DWH's data model and meeting set quality requirements, before finally loading the data into the DWH, where it can be further analyzed and accessed by other (BI) applications [KC04, p. XXI]. Overall, an ETL

[8] Nowadays, data can be procured, just as many physical products, online on *data marketplaces*, cf. [SSV13].

process can become a resource-intensive, time-consuming, and complex. However, it is necessary to actually exploit the benefits of DWHs over simple extract processing approaches [Inm05, p. 18]. According to Inmon, this process can take up to 80 % of development resources [Inm05, p. 287], which underlines the intensity and importance of ETL processes. Consequently, Inmon summarizes that ETL "opened up the floodgates" for proper BI with its inception in 1990 [Inm05, p. 402].

In the *Extraction* step, the main concern is to extract data for integration into a DWH. As source data can be diverse, several aspects of it and the target DWH need to be taken into account when preparing access to these source systems [KR13, p. 55]. It is not just that an underlying database or DBMS is logically different from a DWH schema, also access protocols (both directly with the database as well as at system and network level), as well as the underlying infrastructure (hardware and software) need to be considered. Discovering the "right" source systems that support the desired business decision is also part of this step [KC04, p. 63]. This is followed by logically mapping source and DWH target data. *Data profiling* is conducted to assess the structure and content characteristics, such as completeness and relationships, of the source data [KC04, pp. 68ff.]. This includes detecting anomalies (e. g., unexpected **NULL** values) and eventually assessing data quality to fine tune necessary quality enhancing steps later during transformation [KC04, p. 75]. Profiling can be supported with dedicated profiling tools and approaches [KC04, p. 57]. Kimball points out that it is crucial that a project can be stopped or readjusted at this planning step, if the underlying data sources are unable to support the business goals of a DWH [KC04, p. 57]. Physically accessing source systems means dealing with numerous access methods, such as Java Database Connectivity (JDBC) or Open Database Connectivity (ODBC) for physical RDBMS access [KC04, pp. 76f.]. However, for complex Enterprise Resource Planning (ERP) systems as SAP BW, Kimball advises against a direct database extraction and recommends dedicated ETL connectors to access underlying data [KC04, pp. 102]. Another point to decide is, depending on the ETL schedule, if an extraction program actively accesses a source, or if the source itself propagates data to a defined endpoint such

an File Transfer Protocol (FTP) server, network drive, or service Uniform Resource Indicator (URI) (*push* vs. *pull* mechanics) [KC04, pp. 261]. More specific tasks to access various source systems are described in detail in [KC04, pp. 55ff.] and are not further elaborated upon here.

The second *Transformation* step can be divided into two major sub-tasks, cleansing and integration [KC04, p. xxiv]. *Data Cleansing*[9] is conducted to ensure high data quality and consistency in a DWH. Cleansing can be compared to the physical act of cleaning dirty surfaces, where the surface should be free from undesired elements such as dirt and in its optimal form afterwards. In an ETL context, cleansing is concerned with correcting or completing the overall data in case of erroneous or missing values, checking consistency and validity both in terms of database constraints (e. g., regarding data type, nullability, or referential integrity) as well as regarding business rules (e. g., matching ZIP code and city) [KC04, pp. 125f.; pp. 132ff.]. Cleansing tasks also build on meta-data gathered during the previous step as result of data profiling. Kimball proposes a structured approach to capture errors during ETL and to audit data comprehensively (cf. [KC04, pp. 125ff.]). Following that, the data needs to be made physically and semantically conformant to the integrated data model.

Integration entails that the various "differences" in the source data must be overcome. For instance, a street and house number may be stored in one VARCHAR field or as two separate fields (one VARCHAR and one INT). There can also be more fundamental structural differences, which need to be resolved when building a DWH [KC04, p. 148]. The reason behind this is that sources databases are aligned to the needs of the specific source application. One database schema may impose different referential integrity constraints on a similar business object than another schema in a different data source, e. g., one may allow for **NULL** values, while the other does not. Moreover, there can be errors and inconsistencies in the data. In addition to spelling mistakes (e. g., "Davdi" instead of "David") or differences in spelling (e. g., "Main street" and "Main Str." denoting the same physical street name), inconsistencies introduce contradictions (e. g., two data sets with different

[9] The term "Data Cleaning" can be and is used synonymously, e. g., cf. [KC04, pp. 113;116]

street names for the same person). In such cases, it has to be decided which variant is the "truth" the DWH should contain [KC04, pp. 73-76; 148ff.]. Further, duplicate data must be detected and dealt with, even after it has been standardized before. The process of deduplication matches certain records using a similarity-based score and removes duplicate ones [KC04, pp. 156f.]. Kimball notes that deciding on sufficient similarity for two records is not a trivial matter and solutions must be designed individually per application. This includes criteria, which amount of similarity renders a record a "duplicate" one [KC04, p. 158]. Deduplication has been a distinct topic for both practitioners and research alike for the previous decades. Nowadays, it is not just applicable to structured relational data, but to other data sources such as semi-structured data (e. g., JSON), too. For instance, Draisbach and Naumann propose an extensible deduplication toolkit, which uses three types of similarity criteria for matching (data structure, data content, and combinations of the two) [DN10].

An ETL process should also execute more technical tasks, which go beyond the semantics and logical conversion as well as actual merging of data. Firstly, data may need to be converted from DBMS into another. Even when all involved systems are relational, there are still differences in how specific details are implemented, such as the software-specific dialect or feature level of SQL in the RDBMS. Moreover, data types (domains) may need to be checked and harmonized even if they seem similar. Lastly, time values need to be added to the data, where necessary, to render the corresponding data set time-variant (cf. [ISN08, p. 12]).

The final step is the actual *Loading* of data into a Warehouse. The extracted, cleansed, and integrated data is written into the target tables. When it is stored there, this DWH data is available to users and can be consumed by them. This is either done directly or with so-called data marts, which contain subject-specific subsets of all Warehouse data and are detailed further below. [KC04, pp. 159f.]. Because ETL processes are not just executed once and new sources may be added, it must be decided what kind of loading or update strategy is used. One the one hand, it is possible to always load the complete set of all transformed data from sources. This is useful, when data is first

populated into a Warehouse or when new sources are added. For regularly feeding new data into a DWH, delta updates are appropriate, which only contain information from sources into a DWH, which is new, deleted, or modified. However, the process of detecting these changes properly is not always straight-forward (cf. [KC04, pp. 204ff.]).

ETL processes are nowadays supported by various software tools available on the market, both commercial (e. g., Informatica PowerCenter[10]) or open-source (e. g., Pentaho Data Integration[11]) ones. They are able to perform almost arbitrary transformations, eliminating the need to write overly similar transformation programs manually [Inm05, p. 111]. Also, ETL process flows can be designed visualized without the need for knowledge about a specific programming language. Depending on the use case, custom-written transformation programs, can still be used instead or in support of tool suites [KC04, pp. 11f.].

The execution environment of an ETL process is also called *staging area* (as data goes through different steps, or stages, during ETL before it is ready for loading; e. g., cf. [KC04, p. 16]). This is the area data from source systems enters before it is loaded into a Warehouse. It is located between those two. Here, extraction and transformation tasks are performed. The staging area is the source component for a DWH when it comes to loading new data into it. A staging area can have its dedicated storage used for intermediate storage before, during, and after transformation. This can be any form of appropriate storage for the inbound or (partly) transformed data, be it an RDBMS or file system storage (cf. [KC04, p. 16]). However, persistence is optional and only employed as needed. It is possible to conduct some or all tasks in-memory only (e. g., when using a dedicated ETL tool). In this case, the staging area would be virtual. The staging area is not meant to be accessed by users of a DWH [KC04, p. 17].

While the specific order outlined here (ETL) is the most typical case, alternative variations of ETL processes emerged. Data can be, for instance, loaded first and then transformed. This is known as Extract-Load-Transform

[10] https://www.informatica.com/products/data-integration/powercenter.html.
[11] http://www.pentaho.com/product/data-integration.

(ELT). This allows to apply complex transformations directly into the data, possibly using database functions directly without an ETL tool. However, this may be problematic if data is accessed by a user or an application before it is transformed, as data leaves the staging area (cf. [ISN08, p. 227]). Alternatively, the transformation step can be split into two parts. One part is done before data is loaded and the final part afterwards. This is called Extract-Transform-Load-Transform (ETLT).

Traditionally, ETL processes are regularly executed in batches. This means that periodically, an ETL process is started and updates a DWH with all data from its sources since previous ETL execution. Typical periods range from several times a day, to daily, or even less often (e. g., every two to three days or weekly). Although this entails that the most recent data cannot be part of DWH analyses, it allows to conduct effective transformations that focus on quality without harsh time constraints [ISN08, pp. 217f.]. The alternative to this is *online* or *real-time ETL*. Just as with online transactions, data should be made available to a DWH without additional delays. In real-time scenarios, even more strict temporal restrictions can be applied. In such cases, an ETL process is performed directly after the execution of a transaction in the source systems or an ETL process is triggered by pushed data from a source. However, as Inmon notes, strict time constraints may adversely affect required quality levels in a Warehouse. Real-time ETL leaves less time to conduct complex transformations and cleansing activities to enhance data quality [ISN08, pp. 216f.]. Nowadays, both types of ETL execution are technologically feasible. Thus, the decision which of the forms to pursue, should made based on business requirements [ISN08, p. 218].

OLAP, Multidimensional Data Models, and Data Marts

OLAP using a Data Warehouse builds on data that is stored in a *multidimensional data model*. It makes use of relationships specified in, e. g., relational data models in source databases of a DWH. These models are built around the concepts of *facts* (e. g., a business sale) which are analyzed with *measures* (e. g., sales revenue) along *dimensions* (e. g., product, time, or region) [EN16, p. 1108].

Multidimensional models can be visualized as data cubes, which are essentially multidimensional matrices. These are called *OLAP cubes* when exceeding two dimensions of analysis in an OLAP context. Elmasri and Navathe explain that multidimensional modeling improves query performance in comparison to relational models when working with data cubes. [EN16, pp. 1105ff.]

The purpose of such multidimensional models it to analyze data using various dimensions associated with it. For instance, in addition to a total revenue for products of an organization, it might be interesting to see the revenue by year. This results in a two-dimensional view (i. e., a table or spreadsheet). When another dimension, such as a sales region, is added, a three-dimensional data cube is created[12]. Figure 2.2 visualizes this example of sales revenue with a temporal, spatial (i. e., regional), and product dimension.

The multidimensional model is represented by so-called *fact tables* and *dimension tables*. Facts in fact tables are represented by one tuple (and, thus, by one row) and pinpoint the variables that are to be analytically measured. For instance, a sales fact contains the revenue and costs of sales which are to be further analyzed. Facts are identified by one or more dimensions. A sales fact might be identified by the involved customer, the bought product, and the time of sale. Each of these dimensions is stored in a separate dimension table and is pointed to by the fact table via a foreign key relationship. Elmasri and Navathe point out that these dimensions often resemble the "master data" of the transactions, which a DWH's operational sources work with. Two forms of organizations for this exist. In a *star schema*, each dimension resides in a single table. If a normalized dimensional hierarchy were to yield several tables, these would be denormalized to end up with one table only. During this process, redundant data and violation of normal forms is accepted to avoid costly joins between dimensional tables. In a *snowflake schema* dimensions stay or are deliberately normalized. Figure 2.3 illustrates the three-dimensional model behind the example cube from Figure 2.2 in

[12] The term *hyper-cube* can be used for cubes with more than three dimensions [EN16, p. 1105].

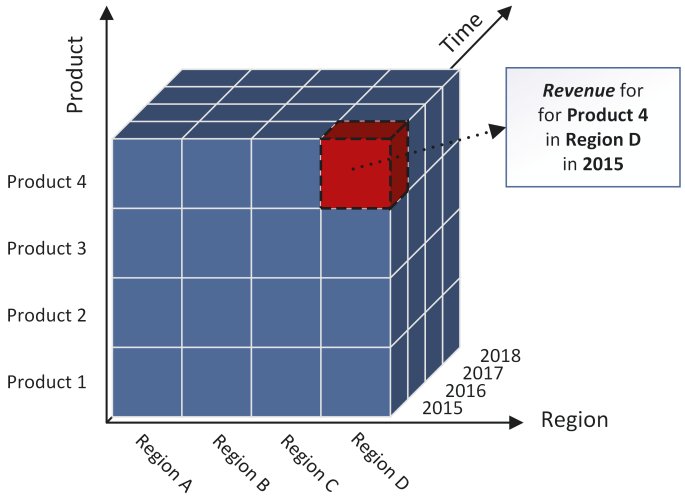

Figure 2.2: Example for a three-dimensional OLAP cube with sales revenue facts along the dimensions *Product*, *Region*, and *Time* in years.

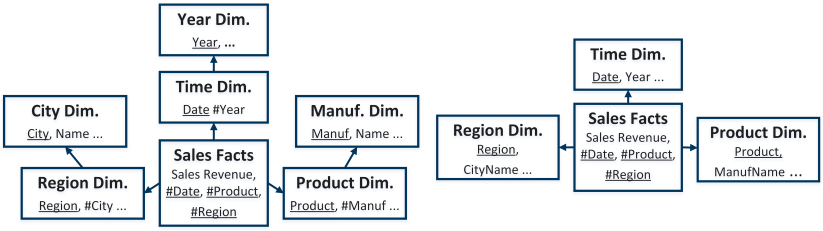

Figure 2.3: Forms of multidimensional data organization for DWH data for OLAP analysis, illustrated on example sales data. Left: *Snowflake schema* with normalized dimensional hierarchy; Right: *Star schema* with denormalized dimensional hierarchy and attributes condensed in one table per dimension.

both variants. A further variant of these is a "(fact) constellation schema", where several fact tables share dimensional tables [EN16, pp. 1109f.].

To change the view of cube as the one in Figure 2.2 *pivoting* and *slicing* can be utilized. Pivoting resembles a rotation of the cube to switch the perspective of the data. In this example, the cube could be rotated so that the *Region* dimension represents the rows on the left, and *Product* dimensions the columns on the Y-axis, with *Time* remaining on the Z-axis. Slicing reduces a multidimensional cube to a two-dimensional spreadsheet view. [EN16, p. 1106]

Two other important operations on OLAP cubes are *drill-down* and *roll-up*, which build on dimensional hierarchies in a data model. When drilling down, the level of granularity of a dimension is increased, revealing more focused data. In the example case in Figure 2.2, a drill-down on the *Time* dimension would break down years into quarters, which could be further broken down into months or days. So, instead of seeing the revenues for the year 2014 along the product and customer dimensions, one might view these for the 1st to 4th quarter in that year instead. This allows to see aggregate data on a more detailed level. This can be done similarly to other dimensions, where a respective hierarchy is defined in the data model, e. g., for regions and other spatial data. A *roll-up* is the inverse of the operation. It decreases the

level of granularity regarding aggregate analysis of facts. If quarterly data is displayed by default, a roll-up to a yearly level is possible. [EN16, pp. 1106f.]

Physically, data can be stored directly in multidimensional form in specialized multidimensional databases. This form of OLAP is called Multidimensional OLAP (MOLAP). Such databases and models are usually separate from a regular DWH [EN16, p. 1108]. However, multidimensional star and snowflake schemas can be readily modeled using relational tables. When using these, new SQL functions and operators allow to conduct OLAP analyses on relational data. In this case, the term Relational OLAP (ROLAP) is used. Starting with SQL:1999 and enhanced by SQL:2003, the SQL standard incorporated functions to perform OLAP queries [ISO99] [ISO03]. These added cube operators such as **CUBE** and **ROLLUP**, as well as the generalized **GROUPING SETS**, to be used in conjunction with **GROUP BY**. Using these, aforementioned OLAP cube operations can be directly executed on relational data (cf., e. g., [262, 261]). Other notable additions are window functions, which can execute analytic SQL functions on a window frame of the result set instead of the whole result set. A window frame represents a subset of rows and can be specified with the OVER clause after a column in the **SELECT** statement. In the example cube above in Figure 2.2, a revenue ranking by *year* could be added so that the best-selling product/region combination for each year could be easily identified (cf., e. g., [211]).

Importantly, a normalized relational data model, which is based on the integration of all incoming data sources, can still be advantageous for data in a DWH [Inm05, p. 367], as it allows fine-grained and flexible storage (e. g., for updates to the model) and abstraction of various input data sources [Inm05, p. 363]. In fact, a Data Warehouse can be implemented as an extended RDBMS [CD97], although some features of these may not be needed [Inm05, p. 172]. With so-called *data marts*, which contain a subject-specific structure of DWH data according to the analytical needs of the subject-group [Inm05, p. 370]. Such groups could be marketing or sales departments in an organization, who focus their analyses on a particular subset of the integrated data relevant to them (cf. [Inm05, p. 176]). To fulfill the requirements of such groups, multidimensional models tuned for OLAP are well suited

[Inm05, p. 365]. The reason is that an integrated relational DWH data model is not optimized (e. g., with respect to query performance) for access of these groups [Inm05, p. 365]. Therefore, multidimensional models for OLAP analyses and analytic workloads are an appropriate fit especially for data marts, which are built from a global data stock in a DWH [Inm05, p. 175; p. 363]. On the market, several vendors offer dedicated OLAP server products, designed for multidimensional analysis (e. g., Oracle [Sch08]).

Reference Architecture

The DWH reference architecture, depicted in Figure 2.4, describes a commonly accepted template pattern to build a custom DWH and to summarize common DWH patterns and components. Not including certain implementation details, it is an agreed upon reference architecture. It acknowledges the fact that most DWHs follow a similar structural and behavioral pattern and that they are based on similar assumptions. This reference architecture encompasses typical layers, their relations, and typical components of a DWH. It illustrates the basic flow of data from its origin sources through ETL into an integrated set in the DWH core, from where it can be disseminated to data marts and other BI applications.

Notably, the central component of this reference architecture is also referred to as *Enterprise Data Warehouse* in order to disambiguate earlier approaches to Data Warehousing from the now agreed upon reference architecture and to outline the understanding that a DWH spans the whole enterprise (also in contrast to departmental, less-integrated approaches) (e. g., by Inmon in [ISN08, pp. 11ff.]). In an ideal architecture, the would be a strict separation of layers. However, in reality, the data mart layer is only optionally used. For instance, as outlined above, OLAP can be conducted directly via SQL, which allows to execute it without dedicated data marts directly in a relational DWH. Other applications may also opt to directly access DWH data.

The DWH reference architecture is revisited and put into a greater context when discussing alternative and novel (reference) architectures in Chapter 3.

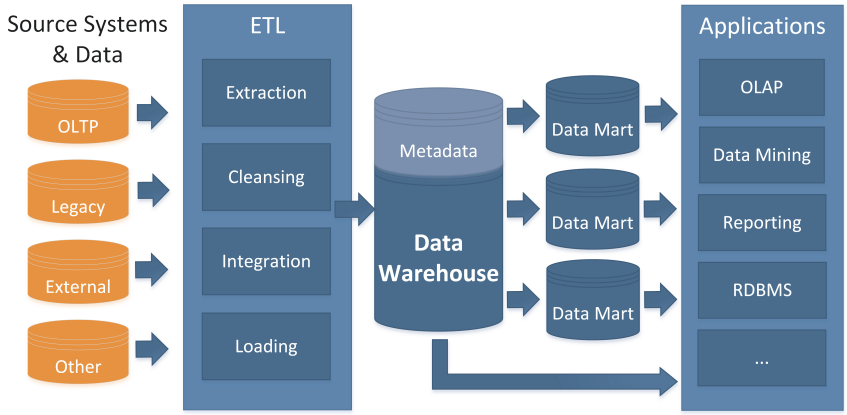

Figure 2.4: Traditional Reference Architecture for a DWH. Based on [EN16, p. 1103], [Inm05, p. 72; pp. 175ff.; pp. 182f.] [ISN08, pp. 11ff.].

Data Mining

Data mining is traditionally associated with Business Intelligence [EN16, p. 1069]. Elmasri and Navathe note that data mining is often simply referred to as *data analytics* [EN16, p. 1069]. While this work uses the more general term data analytics in subsequent chapters, data mining is used here to present it in a BI and DWH context. Data mining, besides OLAP, is one of the typical application scenarios of a DWH. Historically, the "mining" part of data mining is used to stress that it goes beyond "just" analytics. It is to convey that it requires preparatory work, which should be focused, just as in the actual mining process of refining ore to more valuable materials (e. g., transforming iron ore into ingots or into steel) [Agg15, p. 2]. This further compares to value-adding data processing through ETL in a DWH, which enables further analytic usage. Essentially, data mining is the "discovery of new information in terms of patterns or rules from vast amounts of data" [EN16, p. 1069]. In a broader context, Aggarwal sees it as "study of collecting, cleaning, processing, analyzing, and gaining useful insights from data" [Agg15, p. 1]. Data mining cannot only be used to discover

information, but also as means in discovery of *knowledge* based on that information. Hence, it is seen as part of typical processes for Knowledge discovery in databases (KDD). KDD processes themselves are comparable to the process of information generation using ETL in a DWH (cf. [EN16, p. 1070]). Thus, in spite of the earlier definition, it is used to refer to several related research fields. These are nowadays associated with data analytics. On the one hand, there are traditional fields of statistics and mathematical optimization. But there are also modern fields such as machine learning or Artificial Intelligence (AI) [EN16, p. 1099]. Due to its broad definition and related fields, data mining is also effectively an umbrella term, which relates to specific forms of data processing and analytics. More extensive details on data mining and related tasks and techniques can be found in various literature sources such as [Agg15], [WFHP16], or [HKP11].

While data mining is naturally associated with DWHs, it can be conducted separately and directly on operational data as well. However, the integrated and cleansed body of data in a DWH through ETL is especially useful and efficient for several cases of new information discovery [Agg15, p. 3] [EN16, p. 1070]. In a DWH, data mining allows to discover patterns that cannot be discovered by straight-forward querying and processing data with SQL only. Elmasri and Navathe thus advocate that data mining needs should be incorporated into the planning of a DWH [EN16, pp. 1070f.]. For performance considerations, however, it might be advisable to build a separate "exploration warehouse" for data mining based on an existing, ordinary DWH. This exploration warehouse is dedicated to heavy statistical analyses techniques used for data mining, which could otherwise negatively impact regular DWH operations [Inm05, pp. 380f.]. OLAP in itself already offers "limited" data mining capabilities [HKP11, p. 154], which include not just "regular" analytical processing with OLAP cubes and comparatively simple aggregate measures (cf. [HKP11, pp. 153f.]), but also functional models for forecasting and statistical analyses, turning OLAP into a formidable data analytics tool [HKP11, p. 148].

Typical tasks for data mining include [EN16, pp. 1071ff.] [Agg15, pp. 14ff.] mining of association patterns, classification and regression, detection of

outliers, and cluster analysis. Their details are further elaborated upon in Section 2.3.

Data mining generates *inductive knowledge*, i. e., knowledge through new rules and patterns in existing data [EN16, p. 1075]. Its counterpart is deductive knowledge, which uses prescribed rules and pattern to gain new information from data [EN16, p. 1075].

Usage Scenarios

The primary purpose of a DWH is to offer an integrated set of data over all sources according to the defining characteristics outlined above. This stock of data enables a plethora of usage scenarios, especially for analytics and decision-support.

As with RDBMSs, DWHs were initially tuned to ingest textual data, mostly structured data in relational form [Inm05, p. 257]. Nevertheless, other textual data, such as external data in form of websites, flat files, or from services (e. g., in form of JSON or XML data), can be processed and transformed for storage in a Warehouse. However, larger amounts of such data necessitate different approaches (e. g., for data cleansing). The implications of this and other novel pattern of DWH architecture are discussed in Section 3.3.

Typical usage scenarios can be grouped into distinct categories [EN16, p. 1102] [KR13, pp. 22f.; pp. 231ff.]. Three of those were presented here, while other categories are new or contain a specific subset of functionality of the aforementioned. These categories include:

- **OLAP** analyses using data cubes.
- **Data Mining** for inductive knowledge discovery.
- **Reporting**. Reports are founded on querying capabilities of a Warehouse. Reports can include KPIs such as sales figures and other complex analyses. These can be compiled into managerial reports (through *pre-prepared queries* of varying analytical complexity) for decision making. Alternatively, *ad-hoc queries* lead to highly customized reports.

- **Executive Information Systems (EISs)** or Management Information Systems (MISs). These are a class of systems that offer dedicated decision-support for business executives, supported by "appealing" visualizations, such as data charts [Inm05, p. 239]. This includes analyses of trends, KPI reports, internal and external business event monitoring, and OLAP analyses [Inm05, p. 240]. Although EIS are regarded as a distinct IS category, several tasks overlap with DSS, OLAP analyses, and CRM systems. Notably, they are often seen as example for DSS [EN16, p. 1102]. These systems are fully aligned to the vision of BI [Inm05, pp. 239f.].

- **Modeling and Forecasting Tools**. These focus on heavy statistical methods for predictive analytics. Application areas include budget or market forecasts. Methods usually employed for data mining are applicable as well. [KR13, p. 23]

- **CRM Analytics**. Among various application domains, analysis of customers has distinctive qualities, which make it suitable for DWH analyses [KR13, pp. 231f.]. CRM has a "split personality" [KR13, p. 231] between operational and analytic needs. What is denoted as omnichannel strategies or 360-degree views of the customer require an integrated set of data between operational CRM source systems and a deep analysis of customer metrics, which makes this kind of analysis ideal for DWH environments. [KR13, pp. 231.]

A similar categorization can be found in [Leh03, p. 40], where usage scenarios are further evaluated with regard to information complexity and analysis flexibility.

These usage categories, except CRM analysis, are applicable in various organizational domains, such as sales, inventory value chains, order management, accounting, procurement, and human resource management. Additionally, they are applicable to numerous industries such financial services, telecommunications, transportations, education, e-commerce, insurances, or healthcare (cf. [KR13]).

2.2 Big Data Technologies

This section introduces the trend and phenomenon "Big Data" and describes a selection of key Big Data technologies, which are suited to store and process Big Data. Several new Big Data technologies have entered the market in the last decade. Their list is continuously expanding (e. g., cf. the Big Data landscape by Turck [381]). Due to this rapid evolution, an exhaustive enumeration is not purposeful. However, there are several technologies widely discussed in academia and research, which are representative examples of the vast spectrum of Big Data technologies and are presented in the following sections. They represent solutions for the various challenges related to Big Data. First, Section 2.2.1 elaborates upon the term Big Data and its history, discusses its definition, and outlines Big Data characteristics known as "Vs". Next, so-called NoSQL data stores are depicted in Section 2.2.2, which were one of the initial storage technologies related to Big Data and emerged as alternative to RDBMSs. Afterwards, Hadoop Distributed File System (HDFS) in Section 2.2.3 and MapReduce in Section 2.2.4 are discussed in greater as detail as prominent representatives of several tools from the so-called Apache Hadoop ecosystem. Following this, an overview of this extensive Hadoop ecosystem is given in Section 2.2.5. Subsequently, the category of Streaming Processing Technologies is described in Section 2.2.6. Finally, Big Data advancements to traditional storage technology are outlined in the previous section is presented in Section 2.2.7.

2.2.1 Big Data Phenomenon and Characteristics

Big Data has seen widespread interest from both academia and practice during the last decade, starting around the year 2009. However, origins of the term itself can be traced back to the 1990s and beyond. In 1998, John Mashey from Silicon Graphics International (SGI)[13] held a presentation concerning "Big Data". Mashey used the term to denote growing data sizes and relate it to various storage and processing capacities [251]. Steve Lohr from the

[13] http://www.sgi.com, meanwhile purchased by Hewlett Packard Enterprise.

New York Times Magazine traced further mentions back to 1989 and asserts that the foundations for the term to evolve were laid in the field of etymology by the end of the 1970s [239]. Several other mentions of it can be traced back to final decades and years of the 20th century (e. g., [288]). Diebold attributes coining the term academically to himself in 2000 and others around the same time [Die12].

When understanding Big Data simply as data, which is very big in size, indeed several additional discussions can be found to tackle it and related challenges. For instance, a search for academic contributions regarding "handling large data" between 1990 and 2000 on Google Scholar yields thousands of results [189]. This does not only concern the field of computer science and modern IT-related technologies. According to the Oxford English Dictionary, the term *information explosion* was first used in the 1940s, before modern data processing was available. It denotes a "rapid increase in the amount of information available" [319]. Nevertheless, the spread of modern IT not just in larger organizations but also in private households significantly increased the amount of data generated. IT allowed to scale manual processes to a much larger scale. In US banks, 18 billion checks were processed in 1970, after check processing was moved to computers in the 1960s. This number increased to more than 55 billion in 1992 [Cor05, p. 51]. Meanwhile, transactions clearing costs could be reduced by over 80 % from 1990 to 2000 [Cor05, p. 52]. Cortada notes that in the beginnings of electronic data processing, only large enterprises could afford the initial financial expenses for such digital systems. More importantly, they benefited largely from these as well [Cor05, p. 49] and enabled them to match the increasing demands on check processing capacities [Cor05, pp. 44f.]. On an individual level, the term *information overload* is used. Coined first by Toffler in [Tof71], it expresses that an abundance of information hinders a human mind from effective decision making on a given topic. This also stresses the need for IT to keep pace with a large corpus of data available at a given point in history.

It can be concluded that increasing data size has always posed a challenge. However, more organizations both generate and have access to enormous amounts of data nowadays than in the past. In 2000, the overall data volume

stored across the globe was estimated at 800,000 PB [ZED+12]. In 2014, Facebook's Data Warehouse contained more than 300 PB of data [392]. By the end of 2015, the storage capacity for YouTube videos alone was estimated to be around 400 PB [382]. Zikopoulos et al. estimate that global data size will reach 35 Zettabytes (ZB) by 2020. A similar number is mentioned by market research company IDC [IDC14]. Monthly global traffic via Internet (Internet Protocol (IP)) is estimated to exceed 220 PB per month by the same year [Cis17]. In one "Internet minute" in 2017 alone more than 150 million mails are sent, more than 15,000 GIFs are transmitted via Facebook Messenger, and more 3 million searches are conducted on Google's search engine (cf. [160]). In summary, organizations can generate severely more data themselves than before and also have access to an abundantly large corpus of existing and new external data.

A major development that amplified the emergence of Big Data technologies was *Web 2.0*, which marked the second generation of Web applications. Three different development converged here [Vos14, p. 4]. First, rich and highly interactive Web 2.0 applications and services, which are widely used nowadays, were developed. Secondly, several of them allowed users to participate in many ways on these sites. Social media sites like Facebook or Twitter especially focused on user-generated content and not just consuming premeditated media ("prosumers" vs. "consumers") (cf. [VSD17, pp. 35ff.]). Other types of new Web applications include rating sites such as TripAdvisor[14] or Yelp[15]. These sites attracted massive amounts of users (e. g., there are more than two billion social media users [216] [VSD17, p. 49]). These users do not conduct business transactions, but a limited set of operations instead (usually, Create, Read, Update, and Delete (CRUD) operations). Thirdly, these applications are rapidly changing to accommodate users' needs and to fend off competitions. This led to data storage systems, which focused less on transactional as well as consistency guarantees and strictly structured data models, but could scale horizontally (e. g., NoSQL data stores).

[14] https://www.tripadvisor.com.
[15] https://www.yelp.com.

On the hardware side, other technological foundations needed to adapt to this. For decades, Central Processing Units (CPUs) could become quicker by increasing their clock speed (*vertical scaling*, cf. [EN16, p. 845]). But physical limitations in form of heat meant that alternatives to this had to be found. Naturally, more machines could be used conjointly for *distributed processing* (inter-node parallelism). However, apart from that *multi-core* CPUs were introduced in the first decade of the new millennium (e. g., the IBM Power 4 in 2001 [212] or the Intel Pentium D in 2005 for regular consumers [363]), which enabled widespread parallel processing capabilities inside a computer (intra-node parallelism). Likewise, storage of huge amounts of data necessitates a distributed form of storage (e. g., Storage Area Network (SAN)) of it as well, as an increasing number hard drives are needed, because development of hard drive capacities did not keep pace with the increase of data generated (e. g., cf. [166]). Further, increased capacity and decreased cost of Random-access Memory (RAM) (cf. [254]), resurrected previously unfeasible concepts for mainstream adoption such as in-memory storage and processing (e. g., In-memory Databases (IMDBs), cf. [GMS92]). Similarly, it became an alternative to use computing capacities or software remotely "as a service" (SaaS, PaaS, IaaS, cf. [MG11, pp. 2f.]) instead of a self-maintained on-premise solution [VSD17, p. 20]. Cloud computing providers such as Amazon, Google, or Microsoft offer various services (cf. [185]), which can be scaled up and paid according to usage [MG11, p. 2]. This is not limited to generic infrastructure and computing services, but includes business and even more specialized software (e. g., databases as DBaaS, e. g., cf. [LS10]) as well.

These data sizes grew to such an extent that traditional technology is unable to cope with it efficiently. This concerns both storage and processing. Before the term Big Data became popular, concepts for distributed storage and processing, e. g., *distributed databases*, were already known and used for large data volumes [EN16, pp. 841f.]. Based on the idea of these systems from the 1990s and earlier, distinct Big Data technologies were developed, which inherently rely on *horizontal scaling* (i. e., adding more machines to increase computing capacity, cf. [EN16, p. 845]). These were not based existing traditional concepts such as RDBMS, but inherently designed to store

or process huge data volumes [EN16, p. 841]. In contrast to that, distributed DBMS relied on technological assumptions of their decades-old predecessors and had to be altered to fit these new challenges. For instance, rigid locking protocols become far more taxing to performance when coordinated among several computer nodes in a network (cf. [EN16, pp. 858ff.]). Ideally, performance increases linearly or near-linearly with additional nodes so that systems can scale to "big" challenges. Developments around Big Data technologies bring several advancements forward, which can be used for business decision support. That way, Big Data is an enabler for BI.

In 2011, *Big Data* entered the *Gartner Hype Cycle for Emerging Technologies* [181]. It was annotated with "extreme information" and seen as a phenomenon on the rise.

In Gartner's hype cycle, emerging technologies are evaluated through various phases, one of the is the "peak of inflated expectations", the highest hype phase, where more and more is publicly expected from a technology than it can deliver. After a phase of "disillusionment", a technology slowly enters a state of meaningful usage, i. e., productivity.

In 2015, after going through a hype phase and being at a low phase as not all overly high expectations could be met, Big Data left the hype cycle again. Surprisingly, Gartner argued that Big Data recovered quickly from these initial setbacks. According to them, Big Data technologies were established so well in the meantime that they are not considered "emerging technologies" anymore (cf. [402, 380]). This assumption is reinforced by many new Big Data technologies, of which some gained widespread interest since then (e. g., new tools entering the Hadoop ecosystem). Further, other trends building on analysis and usage of Big Data, since appeared separately in the hype cycle, such as deep learning, machine learning, AI [187, 381], or in-memory technology [182][16].

Challenges related to Big Data are not only technological. As Big Data implies new paradigms to working with data (cf. [NIS15c, p. 5]), other areas of an organization are necessarily affected and may need to change.

[16] Gartner even considers several sub-trends in a dedicated hype cycle for "in-memory computing technology" [183].

Considering the Technological, Organizational, Regulatory, and Economic (TORE) analysis framework from Section 2.6.2, there are also organizational, regulatory, and economic challenges in the context of Big Data (cf. [Vos14]). Organizational aspects include the structure of a business, which must be geared to be able to make efficient use of all data. Even when technology is there, several departments might need to work to together and combine different areas of expertise regarding data. On the regulatory side, protection of privacy is widely discussed in the context of Big Data. Economically, great potential of vast amounts of data is met by additional complexity to comprehend a holistic view and approach to leverage it. Notably, this section focuses primarily on the technology aspect of Big Data as foundation.

Definition

Many definitions of Big Data have been proposed. In spite of that, there is no agreed-upon definition yet. Partly, this is due to the multiple facets of Big Data. Naturally, it can refer to data itself. On the other hand, it is used to outline novel technologies (e. g., Hadoop) or methods (e. g., predictive analytics or MapReduce) associated with it. However, what is considered novel changes over time.

Hu et al. analyze and categorize definitions of Big Data into three types [HWCL14, p. 654]. The category of *attributive* definitions captures perceptions of Big Data that refer to properties of it, beyond just size, and associated challenges. *Comparative* definitions see Big Data as "moving target", based on the notion that traditional technology is incapable of dealing with it along each of step of a value chain (e. g., ingestion, storage, or analysis). *Architectural definitions* focus on the underlying architecture consisting of software tools, hardware, and techniques that is specially geared to tackle any form of Big Data.

The US National Institute of Standards and Technology (NIST) performed a non-comprehensive survey on Big Data definitions, which yielded 25 definitions, both from academia and practice [NIS15c, pp. 10f.]. Another informal survey is conducted by Dutcher, where experts were asked to voice their definition of Big Data. Several other facets become

Table 2.2: Selection of exemplary Big Data definitions. Found through: [NIS15c, pp. 10f.], [HWCL14, p. 654], [164].

Source	Definition
NIST [NIS15c, p. 4]	"Big Data refers to the inability of traditional data architectures to efficiently handle the new datasets"
McKinsey [MCB+11]	"Big Data refers to a dataset whose size is beyond the ability of typical database software tools to capture, store, manage, and analyze."
IDC [GR11]	"Big data technologies describe a new generation of technologies and architectures, designed to economically extract value from very large volumes of a wide variety of data, by enabling high-velocity capture, discovery, and/or analysis."
Gartner [184]	"Big data is high-volume, high-velocity and/or high-variety information assets that demand cost-effective, innovative forms of information processing that enable enhanced insight, decision making, and process automation."
Conway [164]	"Big data, which started as a technological innovation in distributed computing, is now a cultural movement by which we continue to discover how humanity interacts with the world — and each other — at large-scale."

visible, such as perceiving Big Data as cultural change beyond technology. A selection of Big Data definitions is listed in Table 2.2.

For this work, Big Data is defined as the ability of data architectures to efficiently acquire, process, analyze, store, use, or manage new data sets. The definition is derived from [NIS15c, p. 4] and amended with specific activities found in Big Data value chains (see Section 3.2) as well as the aspect of manageability as organizational facet outlined in [MCB+11].

It is an architectural definition of Big Data (as per categorization of [HWCL14]), which includes an inherent comparative nature. This definition acknowledges that a data architecture must be capable to deal with size and complexity of Big Data. Nowadays, this often means that traditional data

architectures (e. g., using a centralized RDBMS) are unable to fulfill this task. Nevertheless, as both kinds of technologies, traditional and novel, evolve over time and side by side, they may assimilate properties from the other. For instance, some NoSQL data stores, proposed as a "radical" alternative to RDBMS and SQL, started to introduce typical RDBMS features, such as ACID guarantees. Thus, what is perceived as challenge for established data architectures may change over time. Big Data remains a "moving target". For example, data volumes in 1980s are indeed "small" for todays "traditional" technology (cf. [VSD17, p. 83]).

In addition to this, *attribute definitions* can aid to characterize the actual data behind "Big Data" and what is referred to as "new data sets" in the above definition. Such data can differ severely from traditional structured data (e. g., in an RDBMS or in a DWH). Essentially, Big Data characteristics also go beyond data volume (in contrast to what the name *Big* Data might suggest). Due to this, alternative terms for Big Data have also been proposed, such as "Smart Data" (e. g., [GHP14]) or "All Data" (e. g., [336, 245]), but did not reach the popularity of Big Data.

Characteristics

Big Data is usually characterized through various dimensions. The first three, which were proposed by Doug Laney [Lan01] and gained widespread adoption, are *volume*, *variety*, and *velocity*. These are called "3V". It is common practice to name each dimension with "V" as first letter. The initial three dimensions are agreed upon as common basis. Besides that, more dimensions have been proposed in the meantime. However, there is no consensus about these and a agreed upon set has yet to emerge. Marr and Vossen et al., e. g., consider two additional dimensions, *veracity* and *value* [VSD17, p. 82] [247]. Here, these "5V" are used to characterize all kinds of data, including Big Data. In addition to these, various other characteristic proposals can be identified, such as *variability* (denoting a rate of change in the other dimensions), even up to "10V" (e. g., cf. [NIS15c, vR14, 122, Vos14, VSD17]).

Volume is the name giving property of Big Data refers to potentially huge ("big") data volumes. As outlined above, increasing data sizes and easy access to them, may necessitate a different approach in storing and processing such data. As the NIST notes, global data volume doubles in less than two years [NIS15c, p. 4]. As data volume scales up into PB ranges and beyond, even sophisticated single sever systems are unable to handle it. Distributed storage and processing is necessary to tackle such sizes.

Variety denotes the heterogeneous nature of data (cf. Table 2.3) in terms of data type and format. Previous data analytics efforts and systems focused on structured data (e. g., in relational tables with a predefined data model and structure). Typically, this is transaction related such as from online sales, customer data, or financial data [Mar15, p. 59]. However, in fact most data being generated presently is unstructured (or semi-structured at best) [NIS15c, p. 4]. Unstructured data refers to data where there is no inherent format meta data (e. g., a header). It includes natural text, such as the body of an email, contents of social media posts, but also non-textual data. This can be audio files, e. g. flight recorder data or call records, picture and video files, e. g. from video surveillance or personal pictures on smartphones. Semi-structured data can often be found in Web 2.0 contexts. It includes Web logs, where events (e. g., GET or POST request) on a Web server are noted together with a time in a format, which can be easily analyzed by machine (even without formal header). Other popular formats are JSON and XML, which can represent complex documents. These have numerous attributes that are different or nested for each data item. Such formats do not contain pre-defined fields as structured relational data, but markers so that data elements can be separated by analyzing the data itself [EN16, pp. 426f.] [MCB+11, p. 33] (e. g., by an interpreter program). Textual contents inside semi-structured data (e. g., a "comment" field in a JSON document describing a blog comment) may resemble unstructured content. Most relational databases systems ware not primarily designed to work with such data, although some support may be available (cf. Section 2.1.1). Inmon also points out that DWHs are not well-suited for unstructured non-text formats [ISN08, pp. 34f.]. Notably, variety

Table 2.3: Variety dimension of Big Data with examples.

Data structure	Examples
Structured	Relational data, tables
Semi-structured	JSON, XML, Web logs
Unstructured	Text files and documents (e. g., PDF, social media posts, e-mails), pictures, audio, video

also refers to data being internal or external to the analyzing organization [Mar15, pp. 59ff.].

Velocity is attributed to the speed of data at generation and while "moving around" [246]. Although OLTP already introduced the notion that transactions should be executed without additional delays and respective applications had to cope with many concurrent users, even faster data generation is possible today. For instance, car sensors (e. g., traditional ABS sensors or sophisticated Lidar systems) generate data hundreds or thousands of times per second. This trend is likely to increase with increased number of semi- or fully-autonomous vehicles. For example, a self-driving Google car is said to generate approximately 750 MB of data per second [192]. In addition to that, numerous other Internet of Things (IoT) devices are brought onto the market, which all generated various data (cf. [Mar15, pp. 76f.]). In 2013, more than 100,000 tweets per seconds were sent via Twitter during a special event [235]. On average, there are still 8,000 tweets as of 2018 [216]. Even though traditional forms of data might be generated and processed quickly, they are still in discretized forms with a discernible beginning and end. In contrast to this, streaming data (or *data-in-motion* [NIS15c, pp. 15f.]) represents a large flow of data, where a beginning and an end may be hard to recognize. Even if single data items are small, velocity may lead to huge data volumes over time (a 10 KB data package generated 10,000 times a minute creates more than 1.5 GB data a day). A stream's size and speed prevent it from being stored and processed with ordinary means so that different techniques and technologies are needed (e. g., taking a sub-portion of a stream into account). This can also mean that no means of persistence (i. e., a form of data storage) is used at all

[245]. It also referred to as data-in-motion. Of course, there is still regular data (or in-situ data [PP15] or *data-at-rest* [NIS15c, pp. 13f.]), which can be processed in varying degrees of speed, from occasional or more frequent ETL batches (cf. [CA13, p. 6]) to quick OLTP-style transactions. Besides dealing with velocity in data generation, use cases might require timely storage, processing and analysis of data or even action upon resulting information ("data latency", "analysis latency", "decision latency", cf. [CA13, p. 5]). Stream processing techniques are often employed to meet these *real-time* needs.

Veracity refers to the trustworthiness of data [Vos14, p. 4]. It is determined by its quality and origin, i. e., if the data is dirty or from a (potentially) untrustworthy source. In contrast to internal data, Big Data is often external data (e. g., Facebook or Twitter posts). Neither the platform provider nor the data consumer has full control of data accuracy and quality. These issues of "dirtiness" can be inherent to the data, e. g., spelling mistakes, abbreviations, or colloquial language [VSD17, p. 82] [246]. Some of that might be cleaned up by ETL processes. However, uncertainty can also come up regarding the factual accuracy of the content [VSD17, p. 82]. An Amazon review may be bought by a product manufacturer [403], tweets may be generated by bots to manipulate a political agenda [138], or alleged news story on Facebook may be unintentionally or intentionally misleading or wrong ("fake news") [379]. These aspects need to be taken into account when deriving insights based on such data.

Value indicates potential business value of Big Data [VSD17]. By taking a multitude of additional data sources into consideration, the assumption being that this leads to additional business value through improved insights and decision making [246]. For instance, in omni-channel strategies, all communications on social media of a customer can be taken into account and not just traditional phone or email contacts. However, the value generation process must be adapted to exploit this value. This means that proper a business case backs up a Big Data endeavor (cf. [245]). In traditional (OLTP) systems it is clear that this business data is of high value and thus needs extensive integration and cleaning efforts. Due to the aforementioned characteristics

of Big Data, more tailored approaches for a value chain are necessary to efficiently generate value from data. This is also fueled by the multitude of possibilities, which Big Data potentially *could* enable [247]. Consequently, appropriate measures for value generation have be to taken, depending on both business use case and data at hand. For instance, as long as the business value of a huge volume of data is unclear (e. g., several GBs of Web logs or click tracking data), setting up a complicated and costly ETL process for cleansing and integrating it for future decision making would be premature. Instead, an exploration use case using data mining and a respective environment might be suitable, where data can be separately analyzed more deeply for business insights (e. g., in an exploration warehouse, see Section 2.1.2). Only once positive results are found, data is integrated into a more stable analysis system, e. g., an EIS.

2.2.2 NoSQL Data Stores

NoSQL data stores appeared as reaction to the needs of modern Web 2.0 applications. These did not have strict requirements on data consistency, but needed scalability to cope with many users interacting with those pages. Those interactions and user-generated content on sites like Facebook and Twitter are mostly limited to rather simple operations (CRUD) (cf. Section 2.2.1, [EN16, p. 884]). For instance, updating one's profile, pushing a "like" button, or sharing an image and a message with friends are different from typical business transactions as such actions do not concern critical or high-valued transactions. The content is also less structured or even unstructured (e. g., images or videos, body of custom written text instead of predefined values). As these services evolve, data structures also need to adapt rapidly to accommodate for new features. Traditional RDBMS are not designed for such workloads. The relational model is usually seen as "fixed" and changes to it necessitate updating all existing data in a relation. Transactional guarantees impact performance, while not fully needed for certain kinds of social applications. Complex joins and analytical queries as with OLAP are not needed for these applications and structures that support

these are regarded superfluous. Apart from that, support for large amounts of data is needed ("web-scale" [165]).

The term "NoSQL" itself was first mentioned in 1998 and referred to an RDBMS without SQL interface ("No SQL"). However, only a decade later in 2009, a widespread "NoSQL movement" gained traction [370] alongside Web 2.0 and, later, Big Data (cf. Section 2.2.1). This movement provides an interpretation of NoSQL, which is still used today, namely "Not only SQL". Databases adhering to this principle of exploring other forms of access than SQL and (simpler) alternatives to the relational model are thus called NoSQL data stores. More than hundred NoSQL data stores and other non-relational databases can be found on the market. A dedicated NoSQL website by Edlich lists more than 200 NoSQL data stores [165]. Database engine comparison website db-engines.com also lists more than 150 non-relational databases [218].

Characteristics

Before NoSQL data stores became popular, other alternatives to the relational model were already known. Vossen mentions object-oriented databases for storing objects instead of relations [Vos91, pp. 489ff.] as part of generation of "post-relational" databases [Vos91, p. 8]. Further, XML databases [EN16, pp. 425ff.; p. 888]. Nevertheless, while these also can regarded as NoSQL data stores (cf. [165]), although the newer NoSQL data stores gained more widespread attention and adoption (cf. [218]).

Based on [Cat11, p. 12] and [165], six characteristic properties of NoSQL data stores can be identified:

Scalability. NoSQL data stores should be *horizontally scalable* [Cat11, p. 12]. Instead of increasing speed and capacity of one storage node, more nodes are added to form a distributed cluster (cf. [EN16, p. 845]). Ideally, performance increases linearly. Instead of relying on dedicated appliances, commodity hardware is used in bulk. These off-the-shelf computers available to buy from many different manufacturers.

Schema Flexibility. NoSQL data stores are generally deemed *schema-free* [165]. Schema can be interpreted ad-hoc, if needed. Attributes can thus be added dynamically [Cat11, p. 12]. Elmasri and Navathe describe this as "self-describing" storage scheme [EN16, p. 427].

Access Patterns. Instead of full-fledged SQL, *simple forms of access* such as programming APIs or simplified declarative query languages are used.

Replication and Partitioning. NoSQL data stores employ a shared-nothing architecture [Cat11, p. 12], where — in contrast to replication — no data and memory is shared between nodes (or between processors and cores) [Cat11, p. 13] [EN16, p. 869] so that queries can be distributed between nodes through horizontal partitioning and replication, which provides fault-tolerance as failures are to be expected with many nodes based on commodity hardware [Cat11, p. 12] [VSD17].

Indexing. Cattell notes that NoSQL data stores efficiently use distributed indexing structures and exploit advantages of in-memory storage [Cat11, p. 12].

Instead of offering ACID guarantees, NoSQL data stores often offer less strict consistency to allow for tolerance against partition faults and provide higher availability. This is summarized as *Basically Available, Soft state, Eventually consistent (BASE)* [Cat11, pp. 12f.]. These systems can guarantee *that* they *eventually* reach a consistent state after a transaction has been executed, but it is not guaranteed at which point in time this is the case. For instance, it is not business critical if a Facebook post is only seen a few minutes later on another continent. During that time a local Facebook database and a remote one on that continent would be in an inconsistent ("soft") state, as reads from these different servers would yield different results.

Brewer proposed the so-called *CAP theorem* (Consistency, Availability, Partition tolerance) [Bre00] to classify such trade-offs. *Consistency* is related to its counterpart from ACID and refers to consistency in case of a distributed system, i. e. all clients see the same data [EN16, p. 889] [SF12, p. 59]. *Avail-*

ability denotes the ability of a reachable node to successfully accept read and write requests. *Partition tolerance* (or: Partition-fault tolerance) means that a distributed system continues to function in case of node failures. These cause a partitioning into separated clusters of nodes (hence *partition* tolerance), which cannot communicate [Bre00] [SF12, p. 59] [EN16, p. 889]. Brewer asserted that only two of the three CAP properties can be provided by a (distributed) system [Bre00]. Particularly, consistency cannot be provided under availability and partition tolerance requirements. Brewer's proposal was later reaffirmed by [GL02]. For example, RDBMSs usually exhibit *CA* properties (cf. [EN16, p. 889]). They maintain consistency and availability, while being prone to partition faults (e. g., handling distribution with replication mechanisms) (cf. [EN16, pp. 848ff.; pp. 854ff.]). Aforementioned BASE refers to *AP* properties, while *CP* focuses on distributed consistency and partition tolerance (e. g., when consistency cannot be guaranteed availability is "sacrificed"). Notably, the CAP theorem is built on strict assumptions regarding its properties, so that viable trade-offs between the three properties in practice may be possible, depending on specific requirements on a database or NoSQL data stores (cf. [202, 228]). For example, even in *AP* scenarios, some limited levels of consistencies might be provided (e. g., for transactions limited to a single node).

Data Store Categories

Different categories of NoSQL data stores are available, which can be distinguished by their primary form of data organization. Often named categories include (cf. [EN16, p. 888], [165, 218], [Cat11, pp. 13ff.]) document stores, key-value stores, column-based stores, and graph databases. They are briefly introduced in the following and one exemplary product for each category is presented as category representative.

Document Stores. Document stores organize their content as a collection of so-called documents. A document is a complex object, e. g., in XML or JSON format. For instance, document attributes can not just represent a simple value like a string, but also another nested document or lists of values. While

Table 2.4: Example NoSQL data stores by category including a generic categorization of their CAP property support. This indicates only a general tendency of a default configuration as the CAP properties are not necessarily binary. Examples per system category are ordered by their category's popularity according to the ranking from [217]. Further data on NoSQL data stores not linked here is documented in Section C.1.2. Data based on: [217, Cat11, 210, RW12, SKG+12].

Category	Example	CAP Focus
Document Stores	MongoDB	CP, CA [RW12, p. 319]
	Couchbase	AP, CP [119]
	CouchDB	AP
Key-value Stores	Redis	CA
	Memcached[i]	CP
	Riak KV	AP
	Project Voldemort[ii]	AP
Column-based Stores	Apache Cassandra	AP
	Apache HBase	CP
	Apache Accumulo [iii]	CP
	Google Cloud Bigtable [CDG+08]	CP
Graph Databases	Neo4j	CA

[i] https://memcached.org. [ii] http://project-voldemort.com. [iii] https://accumulo.apache.org.

```
db.RepairJobs.insert({
        _id: 1,
        Car: "BMW 318 Ci",
        Customer: "David Black",
        Cost: 2000.00
})};
db.RepairJobs.find({ Cost: { $lt:
 ↪  3000.00}});
```

Listing 2: Example code for MongoDB. The first line creates collection a "RepairJobs" and adds a row from table *RepairJobs* in Table 2.1 as JSON document. The second line finds all repair jobs, which have a cost value of less than 3000 €.

baring some similarities to relational tuples, documents offer a wider range of values and do not require a pre-defined schema (i. e., fixed set of attribute names and types) for their documents (cf. [Cat11, p. 14]). This implies that each stored document, typically organized by some key, can be unique in its structure with attributes and values. In essence, a document is defined as "self-describing data" [EN16, p. 890].

A popular document store is MongoDB [281] (cf. [218]). It uses JSON to represent documents and a binary variant of it (BSON) for storage [283]. Moreover, it supports creating index structures for its documents to avoid scanning through all of them to satisfy requests. Its API offers basic CRUD operations (see example in Listing 2), including a batch-write operation. To query or find documents, selectors such as "greater than" or "equals" can be used. Besides such logical, array-based, or comparison operators, MongoDB supports regular expressions for matching documents or geospatial operators (e. g., to determine if a coordinate in an attribute is within a specified geographical area). In addition to that, aggregations over documents, even distributed ones, can be computed either using a native pipeline specification, MapReduce jobs, or simple, predefined methods [283]. To scale horizontally, sharding is employed in tandem with replication ("replica sets") to cope

with partition faults automatically [17] and provide availability [EN16, p. 894], but scaling requires additional efforts [RW12, p. 174f.]. Currently, MongoDB guarantees atomicity and "transaction-like" behaviors in the scope of a single document, which a user can rely on, e. g., when implementing a two-phase commit protocol [278]. However, in an upcoming version, MongoDB plans to introduce an option for ACID guarantees over multiple documents by isolating snapshots of data [283]. In spite of this, it is stated that the existing mechanisms in a single document scope would be sufficient for the "majority of applications" [207].

Key-value Stores. Key-value stores feature a very simple data model consisting of simply unique keys and associated values. A key is used as identifier for looking up and fetching an associated value [EN16, p. 896]. Values, depending on implementation of various types, range from simple scalar strings or integer values to more complex semi-structured or unstructured formats (e. g., JSON or binary data). Due to this model simplicity, distributed key-value stores focus on high performance, scalability, and availability [EN16, p. 895]. Elmasri and Navathe state that especially such stores feature rather simple operations via APIs than complex query languages [EN16, pp. 895f.].

For instance, *Redis* [350] is an *in-memory* key-value store. Primarily, it stores data in main memory and only uses secondary storage for data persistence [348][18]. This means that it is optimized for main-memory storage first (i. e., all data is in main memory). While this limits data size to main memory size inside a single node, performance is increased. Thus, Redis advocates itself for write-intensive operations with "small" data sizes per node [348]. This can render Redis a viable choice for caching purposes, even as buffer cache for other NoSQL data stores, which operate on disk or feature more complex data models (e. g., graph databases; see below) [RW12, p. 291; p. 304]. Redis supports not just string values, but also binary data up to 512 MB as key. Values can not only be just simple strings or scalars, but

[17] After a configurable timeout, one of the hidden replicates becomes the new primary node, which answers read and write requests [283]. Their mechanism is based on a master/slave concept [EN16, p. 894].

[18] In-memory databases are also described separately in Section 2.2.7.

also binary data, lists, hashes, or sets [344]. Therefore, Redis is not a "pure" key-value store, but yet does not feature complex data types and structures like a full-fledged document store. By default, Redis was actually originally designated for a single server setup and runs on a single CPU [348] [DT15, p. 148]. It initially featured no automated means for distributed modes, besides asynchronous replication [DT15, p. 146]. On a single machine, multiple databases are best created by starting several Redis instances (cf. [DT15, p. 126]). Although discussing CAP properties usually focuses on distributed systems, a CAP classification of Redis would yield *CA*. Redis Cluster [351], introduced in 2015 [352], uses a master-slave model to implement horizontal scaling up to 1000 nodes [343]. In spite of this, multi-key operations among distributed servers are limited [348, 343]. Moreover, as Da Silva et al. note, Redis Cluster does neither support full *CP* nor *AP*, because trade-offs in both distributed consistency and availability exist. Nevertheless, these trade-offs can be shifted between both sides, depending on configuration [DT15, p. 170] (cf. [343]). An example code for Redis usage on command line is listed in Listing 3.

Column-based Stores. These types of stores are also known as *wide column stores* [EN16, p. 900] or *extensible record stores* [Cat11, p. 14]. What sets these apart from key-value stores is that keys in column-based stores are multidimensional. They may consist of a table name, row key, attribute, and time [EN16]. Cattell describes this data organization as "hybrid" between relational tuples and document as collections of attributes (so-called *column families*) are defined as schema, but each added record can extend an attribute collection dynamically [Cat11, p. 14].

Apache Cassandra [47] is a top-level Apache project and one popular example (cf. [217]) of column-based stores. It focuses on availability and partition tolerance (*AP*) and has the goal to provide scalability and performance [Cat11, p. 20] [Nee15]. Consistency is offered to a certain degree, but even a configurable "strong consistency" is limited in comparison to more consistent data stores (e. g., it is only supported for "lightweight" transactions) [157]. Cassandra column-based data model has certain similarities to a relational model. Its top-level organization is in so-called *keyspaces*, like

```
1  redis> SET RepairJob#1#Car "BMW 318 Ci"
2  "OK"
3
4  redis> RPUSH RepairJob#AllCustomers "David
    ↪  Black" "Gerald Vannen" "Peter Pan" "Peter
    ↪  Kontes"
5  (Integer) 4
6
7  redis> LRANGE RepairJob#AllCustomers 0,1
8  1) "David Black"
9  2) "Gerald Vannen"
```

Listing 3: Example commands for Redis on command line, using values from *RepairJobs* in Table 2.1, which are "flattened" for key-value storage. The first line assigns the value "BMW 318 Ci" to the key "RepairJob#1#Car". The second command on line four appends all customers from *RepairJobs* to a list behind key "RepairJob#AllCustomers", while the last command in line seven retrieves the first two values from aforementioned list. Return values as per documentation [345] are denoted below each command.

databases in relational models. Therein, tables are created. While the naming is equal to relational tables, they are based on the renamed concept of column families as defined above[19]. Each row of data in a column family has a primary key like partition key, which organizes a set of attributes and values under it (cf. [Nee15, 30]). Cassandra features a query language, Cassandra Query Language (CQL), which has a similar syntax as SQL[20], but a more limited feature set and performance caveats when formulating queries (e. g., cf. [172]). Both selections and projections are supported as well as certain use mathematical and built-in aggregate functions[21]. Notably, creating custom indexes on data for faster querying are possible as well[22] and are actually needed for selections (**WHERE**) to work on columns aside primary keys [30, Nee15]. However, joins are not natively supported by Cassandra and have to be handled by application logic [Nee15, 30]. For example, stream processing system Spark with its join-supporting query language SparkSQL (cf. Section 2.2.6) can be used on top of Cassandra data to implement joins [366]. Another possibility is to avoid joins by modifying one's data model (e. g., denormalize data and exploit scalability of Cassandra) [Nee15]. Cassandra based its columnar respective table-like organization on the principles behind Google's *BigTable* distributed storage platform, initially described by Chang et al. in [CDG+08]. Both Apache HBase [69] and Accumulo (cf. Table 2.4) are also inspired by BigTable [CDG+08]. Cassandra's decentralized mode of operations for high availability and eventual consistency are based on a token-ring model proposed by Amazon for their *Dynamo* database (cf. [DHJ+07]), which helps Cassandra to work in an *AP* mode[23]. Additionally, Cassandra offers direct support for integration into Hadoop MapReduce (cf. Section 2.2.4) jobs by providing a data provider, which allows MapReduce to access Cassandra data [Nee15].

[19] In Cassandra 3.0, the naming was changed from column families to tables due its focus on its own query language CQL.

[20] A code sample for Cassandra SQL is omitted due to similarity to SQL for simple queries; cf. Section 2.1.1 for SQL sample commands.

[21] User-defined aggregate functions can also be defined.

[22] Cassandra otherwise automatically creates indexes based on partition keys [Nee15].

[23] E. g., there is no master node [SF12, p. 101].

Graph Databases. In a graph-oriented database, data is represented as a graph. A graph consists of vertices (or nodes) connected by edges (or relationships) [EN16, p. 903]. Relationships between nodes also have a particular direction (one-way or bi-directional) [EN16, p. 904]. Both may contain metadata such as labels for description. This means that associated data can be stored alongside a vertex or an edge [EN16, p. 903]. Graph representation has several advantages. Firstly, analysis of graphs (graph theory) opens various analytics possibilities. For example, in Big Data network analysis (cf. Section 2.3.4), relationships in social networks are algorithmically examined and community structures hidden in graphs are exposed. Notably, Resource Description Framework (RDF) triplets for semantic Web representation exhibit graph structures, so that triple stores for RDF storage are considered graph databases as well, although there are certain differences (cf. [109] [RWE15, pp. 208ff.]). Graphs can also be queried to traverse through their link structure. Graph database *Neo4j* [295] features its own query language, *Cypher*, for finding certain node/relationship pattern in its data [EN16, p. 907]. Neo4j, in contrast to other scalable NoSQL data stores, is not partition tolerant (i. e., *CA*) [296]. It can distribute workload to a certain degree using a master/slave model for read-intensive workloads, but all write operations are still routed through the master node to provide ACID guarantees (cf. [RW12, pp. 250ff.]). Similar to other NoSQL data stores, Neo4j supports a Representational State Transfer (REST) Hypertext Transfer Protocol (HTTP) interface. This allows immediate data access without Cypher queries. Such APIs are characteristic for Web 2.0 applications in general and specifically for NoSQL data stores [RW12, p. 238].

Multi-model Databases and Polyglot Persistence. While not being a distinct category of their own, *multi-model databases* combine several of aforementioned data models, instead of relying on one particular exclusively (e. g., cf. [7]). For example, OrientDB[24] offers support for graph-based data models as well as for document- and key-value-based organization. Here, data models are differentiated at the physical level through abstraction. Based on "records", record IDs, and linkages between records, aforementioned models

[24] https://orientdb.com, recently acquired by SAP [273].

are implemented. Such a record can represent a document, a vertex (node), an object, or binary data. Links can represent graph-based edges (relationships) by accessing bi-directional links between document-wise stored vertices as graphs via a dedicated graph API [127]. A research field fueled by NoSQL data stores focuses on *polyglot persistence*. Polyglot persistence denotes combining characteristics of several DBMSs to fulfill diverse needs of an application, domain or an enterprise [SF12, pp. 123ff.]. Sadalage et al. initially describe this conceptually as counter-proposal to "one size fits all" (cf. [SC05]) RDBMS [SF12, p. 1; pp. 123ff.]. Redmond and Wilson do so similarly [RW12, p. 9]. However, there is ongoing work to unify several databases into one central service. For example [SGR15] proposes a system which can automate polyglot persistence by providing a service and a system, which helps to choose appropriate data models and database systems according to input requirements as well as proposing a mediator services, which resolves queries, operations, and transactions to several DBMSs or NoSQL data stores in the background [SGR15, p. 75].

2.2.3 Hadoop Distributed File System (HDFS)

The Hadoop Distributed File System (HDFS) is, as the name suggests, a *distributed file system*. It focuses on fault-tolerance, availability, and optimizes for throughput performance over access latency [63]. It is part of the Apache Hadoop framework (see Section 2.2.5). The first version of HDFS was released in 2007 alongside other tools as the Hadoop framework [68]. However, the roots of HDFS can be traced back to work on the Apache Lucene search engine, the Google File System (GFS) from 2003 [GGL03], and the Nutch Distributed File System (NDFS) in 2004 [126] (cf. [121]).

In general, a distributed file system uses several server computers called nodes connected via a network to serve files to a client [SGG12, p. 766] [Tan07, p. 603]. Files are spread among the nodes of a distributed file system. To clients, who access a distributed file system, however, it appears no different than regular, centralized file systems, i. e. the access interface enables location transparency. Upon request, a distributed file system locates the file among its network of storage nodes [SGG12, pp. 603f.]. As Silberschatz

ET AL. note, a traditional distributed file system's key performance characteristic is latency, i. e., the time needed to fulfill a client's file request over its nodes [SGG12, p. 603]. HDFS relaxes this, among other standardized distributed file systems requirements, so-called POSIX requirements, to enable "streaming access" to files [63].

Apart from the aforementioned, HDFS is further designated to be run on commodity hardware instead of specialized appliances. This necessitates additional measures to ensure that failing nodes are recovered from efficiently, because failures are likely when hundreds or thousands of nodes with commodity hardware are employed. [63]

HDFS is optimized for *large files* (hundreds of MBs, or several GBs or TBs) that are ideally only written once (i. e., not updated regularly) and read often by (several) accessing applications. Notably, HDFS supports a large number of files, if necessary ("[...] should support tens of millions of files" [63]). Instead of interactive accesses (with small reads or writes back and forth), HDFS is tuned towards batch processing, e. g., when a large portion of the data is accessed for analysis (see also Section 2.2.4). Such usage pattern benefits more from throughput than latency. Written in Java, HDFS can run on all operating systems supported by the Java Virtual Machine (JVM) such as Linux. [63]

This also means that, in contrast to regular file systems (e. g., NTFS or FAT on Windows), HDFS is not run natively by the underlying Operating System (OS). Instead HDFS relies on a "real" file system, on which it is running (i. e., it is essentially executed in user space and not in kernel space, cf. [SGG12, pp. 80ff.]). File system operations and are thus not directly executed by the operating system, but have to be relayed to the HDFS application. HDFS supports a dedicated file system shell, i. e., a command line interface, which accepts various file system commands. They can be directly issued to HDFS using `hdfs dfs <command>` [64]. Alternatively, the Hadoop framework has a dedicated file system shell application, which supports the same set of commands, but to all supported file systems by

Hadoop[25] [62]. The invocation syntax is similar to Unix file system commands (e. g., `hdfs dfs -cp <src> <dest>` to copy a file with HDFS and `cp <src> <dest>` on a Unix shell). Other programmatic access methods such as a C library [61] exist as well.

HDFS Architecture

HDFS is based on a master/slave architecture. The so-called *NameNode* works as master, which clients access. There is only one of them per cluster. The NameNode enforces access rights, manages meta data and the file name space. It is responsible for typical high-level operations with files such as opening, creating, reading, and deleting them. Figure 2.5 illustrates this architecture of HDFS. [63]

To store a file, it is split up into *blocks* (at least one), which all have the same size, except the very last block. A typical block size is 128 MB[26]. Blocks are then replicated among *DataNodes*. Default setting are three replicas per block ("replication factor"). Replication, which is also user-configurable, allows to recover corrupted blocks by copying a replica and other integrity mechanisms (cf. [Whi15, pp. 116f.]). For a 1 GB file, this means that the cluster needs to provide 3 GB of space in total. DataNodes are responsible for physically managing file blocks and replication, when instructed by a NameNode. Which DataNode receives which replica is also determined by the NameNode, but clients can request read and write operations from a DataNode. Typically, one machine in a cluster runs one instance of a DataNode [63]. Moreover, they send periodic *heartbeats* to the NameNode, which contains a block report, informing the NameNode about location and size of a block. This is important, as the location of block could change due to reorganization [EN16, pp. 923f.].

A single NameNode constitutes a single point of failure. While *Secondary NameNodes* can be added, they are not a completely fault-tolerant backup

[25] While HDFS is usually associated with the Hadoop framework, the latter can also interact with the local (native) file system, selected cloud services, HDFS over Web, and others.

[26] This applies to Hadoop 2.0 and newer [Whi15, p. 66]. Version 1 used 64 MB as default [63].

solution to provide high availability. The job of the Secondary NameNode is to merge edit logs into the main file system image, i. e., tracking all changes to the file system. It mirrors this management task of the primary NameNode without being used as NameNode. As there is a delay in updating the file system image on the Secondary NameNode, data loss is likely when the primary NameNode fails [Whi15, p. 67]. A separate initiative, *HDFS high availability (HA)* (cf. [22]), allows to actually run a second NameNode in "hot standby" mode. This means, that this second NameNode can take over from the first primary NameNode without delay [Whi15, p. 69]. This is achieved by having the DataNode send their block reports simultaneously to both NameNodes, primary and hot-standby NameNode. However, both NameNode need to share a the same highly available persistent storage to store their logs. Notably, a dedicated SecondaryName is made redundant by the hot-standby NameNode. Coordination of failovers is conducted by a failover controller based on a modified client library and Apache ZooKeeper (see also Section 2.2.5) [Whi15, pp. 69f.].

Scaling and Performance

In general, HDFS allows to scale distributed storage almost linearly (cf. [Shv10, p. 6]), which is enabled by its principle to separate relatively fast meta data operations (one NameNode) from slower data transfers (DataNodes) [Shv10, pp. 7f.]. Data transfers can be distributed for scale in a cluster without being concerned with other meta data requests [Shv10, p. 8]. However, not all use cases perform optimally in HDFS and new challenges arise when upscaling clusters to a high degree.

The optimization goals of HDFS mean that it is less fit for applications requiring low access latency. Performance may also be hindered by the fact that HDFS needs to adapt to an underlying native file system for storage (cf. [SRC10]).

In addition to this, many small files may be challenging for HDFS as a NameNode holds block information in-memory to increase performance [Whi15, p. 63] [Shv10]. White asserts that millions of files might be feasible (e. g., 1 GB RAM for a "few million" files [Whi15, p. 290]), but billions could

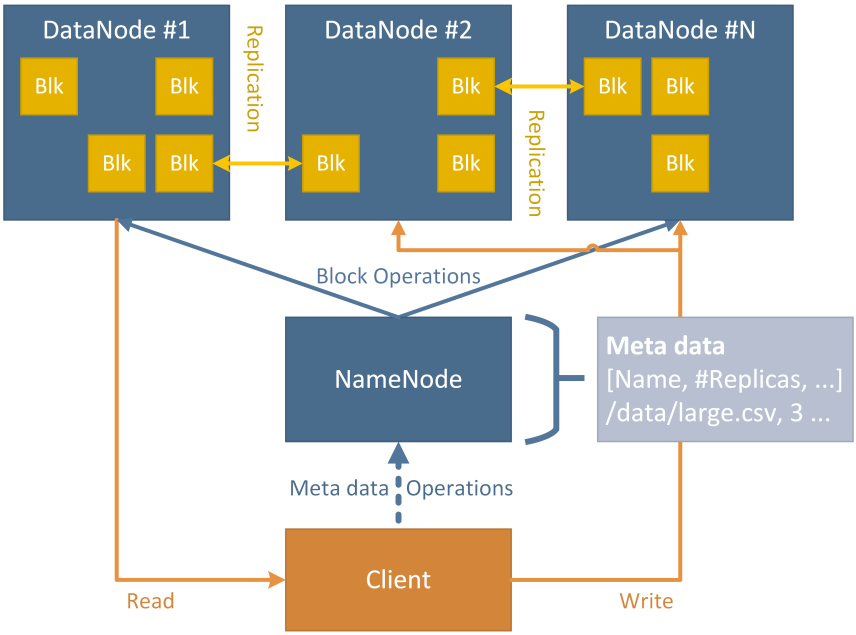

Figure 2.5: Basic HDFS Architecture with one NameNode, three sample DataNodes, including replication of blocks between them, and a client issuing various requests. Based on [63].

not be realized economically with ordinary hardware [Whi15, p. 63]. As of 2010, 100 million files could require up to 60 GB of main memory for management alone, assuming 200 million blocks and 128 MB block size with a replication factor of three [Shv10]. More recently, White calculates that 12 GB RAM can be sufficient to store two million blocks of 128 MB size with replication factor three among 200 nodes, each with 24 TB of storage [Whi15, p. 290]. The actual number of files depends on their size relative to the block size (i. e., at least two billion files). Notably, this applies to the NameNode only, other (Hadoop) applications might require additional memory when installed or run on the same machine as a (Secondary) NameNode. In 2013, O'Dell from Hadoop and Cloud service provider Cloudera recommended between 64 and 512 GB RAM for a "balanced" NameNode server, assuming that 1 GB of main memory allows for 1 million blocks [297]. The actually required amount of RAM may also depend on the specific workload and file structure as well as on the specific version of Hadoop used[27]. Although the estimated numbers differ slightly, a high memory footprint and requirements for a NameNode server remain.

Small files are particularly inefficient as each file takes up space at least equal to the configured block size. A 10 KB sized file would still require 128 MB of actual storage (or the respective HDFS block size value). To reduce the number of files in HDFS, Hadoop Archives (HAR) can be created, which encapsulate several files into one archive file. However, while archived files still appear separately in the file system view, there is no performance benefit (performance may actually decrease as well) when reading archived files using regular Hadoop tools such as MapReduce (e. g., cf. [199]), i. e. only the memory footprint of the NameNode is decreased to a certain degree with this compromise.

To better tackle these main memory related challenges, Hadoop 2.0 added support for *HDFS federation* [Whi15, p. 68]. In such federated environment, the overall HDFS storage space is subdivided into *block pools*, each managed by an independent NameNode (e. g., one NameNode for each top-level

[27] For instance, Hadoop 2.0 improved HDFS performance with various software optimizations (e. g., cf. [395]).

directory in a HDFS deployment). Still, DataNodes communicate with multiple NameNode and store blocks from each NameNode's managed subspace [Whi15, p. 68] (see also [65]). This allows to reduce the number of files per cluster and, thus, memory consumption. For easier client-side management and usage of federated clusters with multiple NameNodes, the *ViewFs* tool can be used [67]. It allows to create customized views on the cluster and "mount" block pools (i. e., their NameNodes) to namespace volumes, which can be directly written to. When writing to it, it is not necessary to know or directly address a specific NameNode or its cluster behind the volume. For instance, a file can be written to */project/file.csv* where */project* is mapped to *viewfs://clusterXYZ/project* in the background. In turn, in the configuration of *clusterXYZ*, each subdirectory on *clusterXYZ* can be mapped to a respective block pool such as *hdfs://name-node-1.clusterXYZ.com:8020/project*. That way, transparency is provided by *clusterXYZ* (cf. [67]).

There are several known implementations using HDFS, which are able to store several Petabytes of data. As of 2014, Yahoo![28] has deployed a Hadoop cluster with more than 450 PB of storage, consisting of more than 4,500 nodes [105]. Facebook's HDFS cluster offered more than 100 PB capacity in 2013 [236]. Shvachko reports that "many" production environments feature 3,000 nodes and 9 PB of storage capacity [Shv10, p. 7]. The Apache Hadoop website lists a multitude of other installation examples including storage capacities in [37]. The specific hardware configuration for nodes in a cluster, besides main memory and storage capacity, depends on the computer workload on these clusters (e. g., cf. [297]).

A particular use case that evolved with HDFS are so-called *data lakes* (alternatively: data reservoirs) [ML16, Inm16]. Data lakes are defined by the notion that all data, both structured and unstructured, are relayed "as-is" without modification and preprocessing. All different kinds of traditional and novel systems, both analytical and operational, can be fed from "the lake" [ML16, p. 179] [Inm16, pp. 14f.]. Several understandings see its foundation as a Hadoop-based system, which uses HDFS as storage. As with NoSQL data stores, there is no prescribed data format. Madera and Laurent de-

[28] https://www.yahoo.com.

scribe a data lake as "methodology to approach [...] raw data" [ML16, p. 178]. While storage is based on HDFS, analysis and decision support are conducted with technology related to Hadoop, especially batch-based MapReduce (see Section 2.2.4) [ML16, p. 178]. Madera and Laurent assess that data lakes actually resemble a new kind of architectural pattern for information architectures (cf. [ML16, p. 176]). There are several possible implications, such as arising data governance challenges [ML16, p. 179], and use cases with a data lake as storage for all incoming data. For the future, they postulate that data lakes should rather resemble a logical view on all data sources in their raw form and should explore other technology. [Inm16] goes into more details about the actual usage patterns of a once established data lake. Inmon argues that a data lake needs to evolve beyond just a one-way storage of data to prevent it from becoming a "garbage dump" [Inm16, pp. 16ff.]. Apart from that, Inmon extensively discusses several options of data lake usage in [Inm16].

2.2.4 MapReduce

MapReduce is a parallel processing programming model first introduced by Google engineers Dean and Ghemawat in 2004 [DG04].

As outlined in the previous section, HDFS enables to store large amounts of data and enable scalability of storage solutions. However, after storing Big Data, it must be processed as well, which is enabled by MapReduce. It is designed to exhibit a similar scalability, i. e., an efficient way to distribute processing power in a distributed storage scenario and leverage parallel processing opportunities. Thus, it is well fit to exploit data stored in HDFS (cf. [121]). A related term is Massive Parallel Processing (MPP).

The initial release of Hadoop 1 already contained Hadoop MapReduce, a MapReduce implementation designed to work on HDFS clusters [68]. HDFS and Hadoop MapReduce are the two initial cornerstones of the Hadoop framework. With the rise of the Big Data trend, these were two of the most prominent representatives of Big Data technology in general. Several new, often Hadoop-related, tools emerged since then. These are presented in Section 2.2.5.

Although the MapReduce programming model was put forth by Google and is used there internally [DG04, 368][29], the Hadoop MapReduce implementation is publicly available and widely discussed. Discussions involving MapReduce refer mostly to Hadoop's MapReduce implementation.

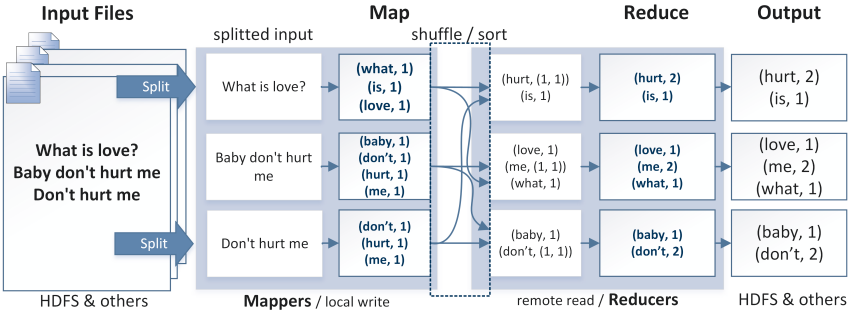

Figure 2.6: Data flow in a typical Hadoop MapReduce application from input files and splitting to mappers, shuffling and sorting, reducers, and final outputs. Technical details like data merging or job preparation and management are omitted. Demonstrated using a word counting example with a song refrain[30]. Based on [DG04], [Whi15, p. 46; pp. 52ff.; pp. 203ff.].

The principle of MapReduce is rather simple and involves its two eponymous phases, *Map* and *Reduce*, aided by support and management activities. [DG04] relies on a "master" program that manages the execution of a submitted user program. The master program is responsible to coordinate parallel execution between the Map and Reduces phases. It allocates the inputs and intermediate results to the respective worker nodes. In Hadoop 2.0 and newer, Apache YARN (Yet Another Resource Negotiator) [60] was introduced and is tasked with resource management and job scheduling and monitoring in Hadoop clusters. These tasks are divided between two sub-components. The *ApplicationMaster* is created for each executed application (or several

[29] Notably, Google since explored and promotes alternatives to the classical MapReduce introduced in 2004, e. g., cf. [252].

of them connected via a graph). It negotiates resources with the global *ResourceManager* and works with worker nodes to execute and monitor the tasks of a job. The aforementioned ResourceManager consists of two components, the *Scheduler* for cluster resource allocations to applications and the *ApplicationsManager*, which is responsible for executing job submissions and coordinating and monitoring it with the worker nodes (similar to the "master" application is described in [DG04]). In contrast to the previous resource management in Hadoop, YARN should scale up to 10,000 nodes and 100,000 tasks [Whi15, p. 102].

The starting point for MapReduce is a problem or workload in form of files, which serve as input. However, these files are split into several parts and distributed to several workers ("Mapper") for the *Map* phase. In the Map phase, a key/value pair is transformed into a list of intermediate key/value pairs. The input keys are typically the document name and the document contents, respectively. For most cases, the input key and values are from different the keys and value emitted by the Map phase, depending on the use case [DG04, p. 2]. For instance, to count words, the key/value pairs produced by the Map phase correspond to "phrase" and "`count`(phrase)" (e. g., "(hurt, 2)", see Figure 2.6.). Afterwards, for the *Reduce* phase, the intermediate key/value pairs, which are persisted on disk, are distributed to Reduce workers ("Reducer"), which read the intermediates from disk. The transfer of mapping intermediates to the reducers is also known as *shuffle*, which is followed by a sorting process[31] before the reducers start to work [Whi15, p. 203]. Here, for each unique of the intermediate key/value pairs, the list of values are grouped (e. g., summarized in the case of integer values). What remains as results are key/value pairs, where there are no duplicate keys and values are agglomeration of the intermediates [DG04, p. 2]. The data flow is illustrated in Figure 2.6.

Distributed processing with MapReduce is designed to be *fault-tolerant*. When using hundreds or thousands of nodes for distributed processing, node failures should be detected and recovered from without users needing

[30] From *Haddaway - What is Love* (1993)

[31] The sorting can also happen in parallel to shuffling [66].

to manually handle such errors. MapReduce works on a *shared-nothing architecture* so that its tasks on Mappers and Reducers are independent from one another. This allows to simply restart or reschedule a failed tasks on a different and have them complete later. [Whi15, p. 31]

Execution of MapReduce exhibits advantages when using a HDFS cluster as file storage for file inputs. First, HDFS has already split files into several large blocks. Thus, no manual splitting needs to be conducted [Whi15, p. 52; p. 228]. This is unless differences between *logical* input splits for MapReduce and *physical* splits in HDFS necessitate a separate logical splitting for a MapReduce job (e. g., cf. [362] [Whi15, p. 229]). More importantly, MapReduce jobs can be directly executed on the DataNodes holding the data to be processed. The paradigm of "moving code to the data" is essential for MPP, when code is smaller than the data is executed on, as it eliminates costly data transfers, which can become prohibitively expensive or even impossible in scenarios with huge data volumes (cf. [Fra12, pp. 91ff.]). With MapReduce, map and reduce code (or sub-programs) can be deployed to ApplicationMasters directly on HDFS DataNodes. Using, e. g., YARN as common management component, resource usage between HDFS operations and MapReduce jobs can be coordinated centrally.

MapReduce applications can be written in Java, the same language Hadoop MapReduce is written in, or other JVM-based languages, but also in other programming languages or script languages. YARN allows to spawn any process on a worker node [285].

To realize the word counting example in Figure 2.6 the following `map` and `reduce` methods could be specified:

Each of those methods is wrapped in a distinct Java class. Besides that, only a job must be instantiated in the program. Specified mapper and reducer classes, input and output paths, as well as other runtime options are set there. Afterwards, the job can be scheduled and run [66].

As it can be seen, MapReduce builds on imperative programming languages. This is different from SQL, which is a declarative language. This means that any algorithm or other processing application needs to exactly specify how the desired results is to be computed from a respective input.

```
1  // private final static IntWritable one = new
   ↪  IntWritable(1);
2  // private Text word = new Text();
3  public void map(Object key, Text value, Context
   ↪  context
4  ) throws IOException, InterruptedException {
5    var itr = new StringTokenizer(value.toString());
6    while (itr.hasMoreTokens()) {
7            word.set(itr.nextToken());
8      context.write(word, one);
9    }
10 }
11
12 // private IntWritable result = new IntWritable();
13 public void reduce(Text key, Iterable<IntWritable>
   ↪  values,
14   Context context) throws IOException,
     ↪  InterruptedException {
15   int sum = 0;
16   for (var val: values) {
17     sum += val.get();
18   }
19   result.set(sum);
20   context.write(key, result);
21 }
```

Listing 4: MapReduce word counting example in Java 10 syntax on Hadoop 3.0.1. Source: Based on [66].

Optimizing the map and reduce code itself regarding input data properties and algorithmic efficiency is the task of a programmer writing a MapReduce application. Naturally, various third-party libraries can be employed instead of writing all code manually, e. g., for data analysis. Since the inception of Hadoop, various patterns for MapReduce applications emerged, which have been documented in literature, e. g., [MS12].

The actual number of mappers ("map tasks") and reducers ("reduce tasks") and their allocation to nodes in a cluster is highly variable. While the above example featured both three mappers and reducers, other variations are possible. Even scenarios without dedicated reducer can be implemented if no Reduce phase is required for the problem at hand [Whi15, p. 55]. A map task is assigned an input split to work on[32]. The Hadoop authors suggest 10 to 100 map tasks per mapper node for a "right level of parallelism". More can be appropriate if a map task is less taxing on the CPU [66]. For instance, given block and input size of 128 MB and an expected total input of 1 TB, there are 8,192 map tasks to be executed for this job [Whi15, p. 222; pp. 228f.]. Importantly, users can not directly influence the number of spawned mappers on a node [66]. If there are more map tasks than can be concurrently executed (tasks usually have memory and CPU requirements, which the resource manager must fulfill when requesting execution containers for them [Whi15, p. 194]), it is waited until these are finished before new map tasks are allocated and run (cf. [Whi15, p. 251]). In contrast to mappers, the number of reducers must be specified by the user, otherwise it will default to only one [Whi15, p. 220]. White asserts that choosing an appropriate number of reducers is more based on experience than on exact estimation. He suggests setting to value so that each reducer runs for approximately five minutes and produces an output equal to the HDFS block size. If tasks sizes are too small, too much time would be wasted on job initialization and management as reducers finish quickly and produce repeated I/O operations for receiving input via network and writing output in small chunks to HDFS [Whi15, p. 222].

[32] Each input split is further broken down into records as key/value pairs. On each of these the user-specified `map()` method is invoked [Whi15, p. 222] [66].

There are many ways to profile and optimize MapReduce execution performance, as several bottlenecks can arise in large-sized clusters. While Hadoop attempts to distribute map tasks so that they are on the DataNode, where their input data resides (data locality optimization), this is not always possible [Whi15, p. 52]. White mentions that most MapReduce jobs are limited by I/O performance rather than CPU capacities [Whi15, p. 181]. Following this, avoiding unnecessary I/O is important. Moreover, persistence of intermediate results (to memory or to local storage) and their transport to reducers is also crucial. An extensive discussion of optimizing MapReduce execution and Hadoop clusters can be found in various sources such as [Whi15]. In total, MapReduce is optimized for *batch processing* workloads. As job setup, scheduling, and execution, including persistence of intermediate results, introduce additional delays, small files or tasks may perform sub-optimally as well. Optimally, MapReduce is served large files (e. g., equivalent or larger than the HDFS block size) (cf. [Whi15, p. 181; pp. 229ff.]).

Besides writing a MapReduce job directly, there are alternatives ways for specifying them. For instance, Apache Hive[33] (see also Section 2.2.5, a distributed Data Warehouse software, which heavily builds on MapReduce. It features partial SQL support [39], but only limited transaction support and ACID guarantees [39]. These queries are typically translated to MapReduce programs and transparently executed in the background by a Hadoop cluster. Notably, other execution environments are supported as well (cf. [88]). Google introduced *Google Cloud Dataflow*[34] as means to model batch (e. g., based on MapReduce) or streaming workflows that are executed in separate Google cloud infrastructure (instead of executing it on on-premise). Their motivation was, among others, to simplify and unify definition of data processing pipelines (cf. [KG15]). More recently, Apache Beam [45] introduces a "unified programming model", based on Google Cloud Dataflow's pipeline model, which is sets out to subsume various programming models (for both streaming and batch) and execution engines under a common data flow pipeline model [84]. Users can design Beam pipelines to solve data

[33] http://hive.apache.org/.
[34] https://cloud.google.com/dataflow/.

processing and manipulation tasks. They can choose to execute them on various "runners", of which MapReduce is one of several options. Naturally, not every Beam feature is supported by every runner (cf. [80]). In particular, this might alleviate issues when defining complex MapReduce jobs (i. e., several MapReduce jobs that need to run in succession) that require manual coordination of the results or might help to avoid to manually define data joins (cf. [MW15, pp. 99f.])[35].

2.2.5 Hadoop Ecosystem

HDFS and MapReduce are traditionally and historically associated with the name Hadoop. Together with YARN and a common Hadoop library [103] they resemble what is known as "Hadoop Core" (cf. [135]). What is now called the Hadoop ecosystem does not only consist of these. Several new technologies, mainly those under the umbrella of the Apache foundation, can be found there as extensions of Hadoop and are related to it, e. g. Apache Cassandra as NoSQL data store (cf. Section 2.2.2 [103]. However, what is now called the Hadoop ecosystem, begins to evolve beyond these core components and their immediately neighboring extensions. Under the umbrella term "beyond Hadoop" (e. g., cf. [Mon13]), technologies emerged, which especially diverge from initial batch processing and strict NoSQL paradigms and form an extended Hadoop ecosystem. More specifically, these include technologies for stream processing (cf. Section 2.2.6) and NewSQL databases as evolution to traditional SQL DBMSs (cf. Section 2.2.7).

Commercial vendors also offer so-called *Hadoop distributions*, which contain multiple technologies from the Hadoop ecosystem and at times also vendor-specific software. In 2016, Forrester Research identifies Cloudera[36] [136], Hortonworks[37] [208], and MapR[38] [243] among the five leading

[35] In contrast to MapReduce, join optimization is a widely researched topic for relational databases. In spite of this, it is done transparently in the background as a user only specifies an SQL statement.

[36] https://www.cloudera.com.

[37] https://www.hortonworks.com.

[38] https://www.mapr.com.

providers[39] of said distributions [125]. Their ecosystem maps give initial hints of core and extended Hadoop components. Further, Roman proposes an ecosystem table, which attempts to list Hadoop-related projects, focused on free and open source software [354].

Categories and tools of the extended Hadoop ecosystem, thus, include:

Distributed Filesystems. This mainly considers *HDFS* as most prominent and often mentioned choice (cf. Section 2.2.3). Roman lists further alternatives in [354].

Distributed Processing. Besides *MapReduce* (cf. Section 2.2.4), there are novel frameworks for processing high velocity streams of data, such as *Apache Spark* with *Spark Streaming*, *Apache Flink*, or *Apache Storm*. Some of these are not only suitable for streaming data, but at the same time for batch data as well, rendering them "universal" processing tools (e. g., Apache Spark). A selection of stream processing technologies is further discussed in Section 2.2.6.

Data Acquisition. Technology for data acquisition is responsible to connect sources of data to various components of the Hadoop ecosystem such as MapReduce for processing or HDFS for storage. To ingest streaming data and control message flow, Apache Kafka can be used. Apache Flume is suited for more traditional sources such as Web logs. Both are detailed in Section 2.2.6. Additionally, *Apache Squoop* [99] is a notable tool. It is designed to transfer data from structured storage such as relational databases or FTP in batch into Apache Hadoop (HDFS) and it also features connectors to aforementioned Kafka [41].

Resource Management. Hadoop unified scheduling and management of compute resources inside a cluster into a dedicated software, *Apache YARN* (cf. Section 2.2.4). Although some Apache projects can run with standalone job management, YARN is regarded a common infrastructure component in the Hadoop ecosystem for this task (e. g., cf. [208]). A newer alternative

[39] Two other providers, IBM and Pivotal have since ceased operations of Hadoop distributions [2].

to YARN is *Apache Mesos*[40]. In contrast to YARN, it allows frameworks using its services to guide its scheduling decisions and renegotiate resource allocations with the goal of better alignment to application requirements (e. g., constantly stream analysis jobs, which YARN was initially not designed for) [359].

Coordination and Communication. In a large distributed cluster, certain coordination and communication task need to be executed repeatably. For instance, common configuration settings should be deployed to all nodes. *Apache ZooKeeper*[41] is a distributed system, designed to reliably communicate information through a cluster, such as configuration settings of MapReduce or other storage or processing nodes. Besides that, *Apache Oozie*[42] is a tool for management and scheduling for DAG-based workflows (which resemble a data and transformation pipeline) in Hadoop clusters. It supports, e. g., HDFS and MapReduce and other related tools such as Apache Pig (see below). Oozie workflows can, for instance, be used to automate ETL workflows in the Hadoop ecosystem [355].

NoSQL data stores and NewSQL databases. Several Apache projects, such as Apache Cassandra or HBase (cf. Section 2.2.2), offer alternatives to traditional RDBMSs. However, other NoSQL data stores can also be regarded as part of an extended Hadoop ecosystem as they represent solutions for Big Data challenges.

SQL + Hadoop. Several efforts are made to complement MapReduce and other imperative programming models with capabilities for declarative queries in an SQL-compliant or SQL-like manner. *Apache Hive*[43] [TSA+10] is designed as distributed Data Warehouse, which uses SQL-like[44] (HiveQL) queries to execute them using MapReduce jobs on underlying HDFS data by "projecting" structures onto already stored data. Later versions of Hive are

[40] http://mesos.apache.org.
[41] https://zookeeper.apache.org.
[42] http://oozie.apache.org.
[43] https://hive.apache.org.
[44] Notably, although there are limits, several SQL features are fully supported [38].

also capable of using Apache Spark as job runners as well as using Apache HBase (cf. Section 2.2.2) as underlying storage [87]. Another notable tool is *Apache Pig*[45]. It offers a high-level scripting language "Pig Latin", which is transparently executed as a sequence of MapReduce jobs on a Hadoop cluster. Pig Latin is positioned as data flow language in-between declarative SQL and low-level imperative programming models like MapReduce [ORS+08].

Machine Learning. Apache Mahout [75] provides machine learning capabilities (cf. Section 2.3.2) on Hadoop clusters (e. g., [32]). Initially, MapReduce jobs were used to implement analytical and statistical functions and model building routines. This meant certain performance trade-offs, especially for online learning (cf. Section 2.2.4). Later versions of Mahout also support Apache Spark (cf. Section 2.2.6), which renders them more flexible for certain workloads (e. g., streaming data) [16]. In addition to that, Spark's MLib [91] offers a set of machine learning functionalities built into Apache Spark, which can be employed without introducing a second software platform like Apache Mahout, as long as MLib alone can meet analytic requirements.

2.2.6 Stream Processing

Stream processing constitutes a new paradigm to analyze high-velocity streaming data. As outlined in Section 2.2.1, such data leads to several challenges. Due to its high velocity, it can amount to large volumes [HWCL14, p. 656]. Moreover, as there is no definitive end to streams, processing and analysis needs to adapt, because regular methods of computation assume a finite set of data-at-rest [HWCL14, p. 656]. For instance, when summarizing a stream of data, it has to be decided when and how often an intermediate result should be reported to users (cf. [HWCL14, p. 671]). Iterative algorithms are specifically designed to cope with such situations. Besides that, streaming data can be used for real-time analysis. Stream processing systems are purposefully designed to process this kind of high-velocity and high-volume data without additional delays, which makes them highly suitable for real-time processing and analysis. For example, one possibility is

[45] https://pig.apache.org.

to only approximate calculations to achieve results quickly (cf. [HWCL14, p. 656]). Streaming applications in general rely on the notion that "fresh" data is more valuable [HWCL14, p. 656]. Thus, a timely analysis is indeed needed for them. In the following, notions of latency and aspects of real-time systems in the context of streaming system are analyzed. Afterwards, typical components of a streaming architecture are described. Next, characteristics of technology for stream processing are outlined. Lastly, an overview of contemporary technologies is presented and described according to aforementioned characteristics.

Latency Requirements

When considering the way of data from the point of its generation to its analysis and usage (see also Section 3.2), several types of latencies can be identified. A latency is the delay time between "action and reaction" [CA13, p. 5]. Depending on the specific application, different types of latency can be relevant. For instance, if high-velocity data must not be immediately processed and analyzed, portions of it might be saved into a regular DBMS and these excerpts can later be analyzed using traditional tools (cf. [SÇZ05]). Nevertheless, most streaming applications do not rely on dedicated storage, unlike most batch-processing systems [HWCL14, p. 656]. Psaltis states that streaming systems make their data available to consuming clients when they need it, while (hard) real-time systems have more constrained needs regarding in-time-delivery and consumption [Psa17, p. 7].

Based on work from [200], Cundus and Alt identify several types of latencies for real-time systems. They can be summarized as:

Data Latency describes the delay between the original generation of data (e. g., an event or a business transaction; so-called *event time*) and its actual availability to the processing system analyzing it, the so-called *stream time*. In stream processing, the difference between these two times is termed *skew* [Psa17, p. 82]. For example, fitness trackers might send their data on periodically in batch to conserve energy, even though they still monitor constantly. Practically, data latency covers the process of data acquisition.

From system side, physical access to a streaming source is relevant, e. g., if data push or pull mechanisms are used (cf. Section 3.2). Then, data needs to be made available by a processing pipeline or a receiving DBMS (cf. [SÇZ05]), after which streaming data is available for further processing. [CA13, p. 5]

Analysis Latency denotes the delay between availability of data to the system and the availability of the analyzed or processed results. Once this step is complete, data can be used for decision making, either by humans or by machines. Technically, this involves the time for pre-processing data as needed, analyzing it, and making the results available for further use e. g., as response to a query[46] [CA13, p. 5; p. 8]). This might also involve accessing other data sources. For instance, to detect anomalies in a data stream, an existing corpus of reference data might be consulted as means of comparison. [CA13, p. 5]

Decision Latency is the delay between the availability of analysis results and actual decisions based on the former [CA13, p. 5]. In prescriptive analysis systems, where the decision is made or prepared inside a system's scope or even executed automatically, technical or processing delays are added to the overall latency. This might involve organizations and humans acting upon received information. For instance, a ground proximity warning system [117] in a plane alerts pilots if they are on a collision course towards the ground and advises them to pull their plane up (e. g., "Caution, terrain! Pull up" via audio). However, pilots must still act manually and in time.

Organizational Latency is seen as the time until analysis results remain relevant by Cundius and Alt [CA13, p. 8]. It indicates urgency of utilizing analysis results. It is situated on the requirements side and an organizational aspect. Technically, it is out of immediate scope of a stream processing system. However, ideally, organizational latency is matched by aforementioned latencies. For example, if a system should deliver results almost instantaneously, it should indicate a certain urgency is making use of these results as well (cf. [Psa17, p. 5]).

[46] Thus, this is also termed *response time*.

Real-time applications refer especially to acting upon information without any necessary delays. Notably, these applications have requirements on accumulations of aforementioned latencies. For example, in car assistance systems such as an electronic stability program (ESP) decisions need to be made within milliseconds [SM13, p. 83]. In 2012, new prototype trains from Siemens for the German railway company Deutsche Bahn[47] had a delay of one second for activation of their brakes after initiating a brake command. This was deemed too slow by authorities [365]. Such systems have "hard real-time" requirements [SM13, p. 85], where tolerance to delays is particularly low [Psa17, p. 5]. In other cases, "real-time" can refer to a few seconds instead of milliseconds. Here, the term "soft" real-time is used [Psa17, pp. 4f.]. Even very fast batch-processing solutions may be suitable, e. g., ones using small batches with limited amount of processing. Hu et al. report that this is indeed the case in practice [HWCL14, p. 656]. Conclusively, real-time requirements can be interpreted differently throughout use cases.

Cundius and Alt [CA13] evaluate certain indicators to determine whether a process supported by an information system satisfies real-time, near real-time, or non-real time (e. g., batch-based) requirements. These real-time categories apply to systems processing and storing data as well. This is because such systems are the basis of mentioned business processes and responsible for fulfilling latency requirements. Taking latencies identified by Hackathorn [200] and hard real-time aspects (e. g., [SM13]) into consideration, time spans can be identified that help to identify if a business system should be considered a real-time, near real-time, or non real-time system. These results are summarized in Table 2.5. For instance, "$\leq$ second span" for data latency means that, real-time systems usually have a data latency that is measured in seconds or less (e. g., milli- or microseconds; "hard" real-time systems are not further distinguished here, cf. [Psa17, pp. 4f.]). It is important to note that actual requirements of real-time applications may be lower than the indicated second or even minute spans (e. g., a requirement on decision latency could actually be in seconds rather than in minutes).

[47] https://www.deutschebahn.com.

Table 2.5: Overview of time spans to distinguish real-time, near real-time, and non real-time systems by different latency types. Based on: [CA13], [200], [SM13], [Psa17].

Latency	Real-time	Near real-time	Non real-time
Data Latency	$\leq$ second span	minute to hour span	$\geq$ day span
Analysis Latency	$\leq$ second span	minute span	$\geq$ hour span
Decision Latency	$\leq$ minute span	hour to day span	$\geq$ day span
Organizational Latency	$<$ minute span	minute span	$\geq$ hour span

Streaming Architectures

Streaming architectures typically consist of the following components (cf. [Psa17, Ell14]). Some of them can be optional depending on the specific use case:

Collection. A collection component is responsible for gathering streaming data into one's own system [Psa17, pp. 14ff.]. As streaming sources are diverse, the focus here is on mechanisms through which data is transferred into target systems. Firstly, push vs. pull mechanisms are distinguished. In pull-based approaches, a streaming system actively accesses a streaming source and transfers data to its own realm. In push-based approaches, a streaming source actively distributes its data to its destinations (e. g., specified by an URI). Interaction patterns for this include request-response patterns, where a client asks for data and gets them as response [Psa17, pp. 16ff.], publish-subscribe patterns [Psa17, pp. 20ff.], or stream patterns, where clients receive a continuous data stream upon request [Psa17, pp. 23ff.].

Data Flow. Management of data flow and communication is a crucial task for streaming systems [Ell14, pp. 17f.]. One of its goals is to decouple production of streaming data from its consumption to avoid congestions in a streaming pipeline. This can occur when systems are not separated as consumption may be slower than production, e. g., when the former is computationally intensive [Psa17, pp. 43f.]. Systems for data flow handle com-

munication between layers in a streaming architecture, e. g., between data collection and analysis. These systems can also act as a buffer and provide message queues and brokering to implement different delivery semantics for streaming systems (see below) [Ell14, p. 19].

Analysis. Stream processing and analysis are core functionalities in a streaming system. Activities performed here include computations like performing aggregations, anomaly detections, or transforming and filtering data for storage purposes. Just as MapReduce enables distributed batch processing, modern Stream Processing Engines (SPEs) such as Apache Storm or Spark Streaming use Directed Acyclic Graph (DAG)-based mechanisms to enable computations and moving of streaming data through their pipelines [Ell14, p. 20] [HWCL14, p. 671]. Projects such as Apache Beam (see Section 2.2.4) also allow to define streaming pipelines, which are transparently executed by an underlying SPE. In streaming programming models, very small "batches" (up to tuple-at-a-time processing, cf. [287]) of data are analyzed by long running processes. This avoids long job start and shutdown procedures as in MapReduce, which induce latencies [Ell14, p. 117]. In contrast to the analysis of non-streaming data, queries against streams are usually continuous, i. e., a query is posed once and updates are continuously sent by a streaming system [Psa17].

Storage. Streaming data or portions of it may need to be stored for long-time archival or other purposes, e. g., exploratory analytics with complex data mining methods. Other use cases might push data back to a streaming system or use it for other real-time applications [Psa17, p. 95]. For traditional long term storage, regular RDBMSs or distributed file systems such as HDFS might apply. These are applicable for "non-streaming scenarios" such as batch analysis [Psa17, p. 97]. Nowadays, it is also a viable option to store data in-memory, which allows for lower latencies and higher throughput (cf. Section 2.2.7). These in-memory systems, particularly ones like Redis, can also be used as a buffer cache [Psa17, p. 105]. Other choices can include, e. g., Apache Cassandra or VoltDB and other fast NoSQL data stores [Ell14, p. 33] [Psa17]. These provide indexed data access and can be sufficiently

fast, especially when they are designed specifically to work with flash-based storage such as Solid-state Disks (SSDs) instead of traditional hard drives [Psa17, p. 105ff.].

Delivery. Lastly, streaming analysis results need to be delivered to a client, e. g., a video call application on a mobile device [Ell14, pp. 33f.]. Especially for the latter, established and new Web technologies like the latest HTTP standard[48] and WebSockets[49] can be employed [Ell14, p. 34.]. These are also suitable for dashboards and desktop apps [Psa17, p. 137]. Alternatively, a direct integration into other business systems via APIs is possible [Psa17, pp. 137f.]. For such client applications it is important to maintain state, i. e., resume processing from where it was previously stopped [Psa17, p. 141]. Further, challenges with respect to loss of data need to be considered as well as can be addressed through use of buffers or package acknowledgments [Psa17, pp. 143ff.]. Recent advancements here are the introduction of query languages such as SQL to stream processing applications (cf. [Psa17, pp. 153f.]).

Processing Technology Characteristics

Technology for stream processing, or Stream Processing Technology (SPT), has several properties by which it can be characterized. These are streaming model, availability, scalability, fault-tolerance, latency, windowing, delivery guarantees, and out-of-order processing. These characteristics, which are explained in the following, are based on [Ell14, pp. 24f.], [Psa17, pp. 27ff.; pp. 47ff.; pp. 52ff.; pp. 63ff.; pp. 69ff.; pp. 79ff.], and [Nas16].

[48] https://http2.github.io.
[49] https://tools.ietf.org/html/rfc6455.

Streaming Model. The streaming model is one of the most basic and crucial characteristics of Streaming Processing Technologies (SPTs) as it defines how streaming data in general is approached. This in turn determines potentials and limitations of a technology. Two basic modes of operation can be distinguished [287], [LLD16, p. 2]:

> **Native.** In native streaming models, streaming data is processed at once as it comes in, i. e., one tuple at a time.
>
> **Micro-batching.** In systems using micro-batches, streaming data is collected for a short amount of time to form small batches of data which is then processed at once. This can even be done by systems designed for batch processing. Depending on latency requirements, this approach might still be appropriate to process streaming data. Lopez et al. conclude that micro-batching is suitable for near real-time requirements [LLD16, p. 2].

Availability. High availability is a key feature of streaming systems, especially if they are to satisfy real-time requirements [Ell14, p. 24]. Especially for hard real-time systems like car sensors, system outages are critical for personal safety. Systems for credit card fraud detection might "overlook" fraudulent events, even when they fail only for a brief amount of time. Besides the availability of a system, this may also concern the availability of the data itself. Here, failures inside data flow management can lead to unavailability of data [Ell14, p. 25]. Mechanisms to ensure availability are *distribution* and *replication*. These are also known from distributed MapReduce jobs and replicated blocks inside a HDFS cluster [Ell14, p. 25]. In case of failures, failover mechanisms are triggered, which allow to recover from a failure state eventually. For instance, this is achieved by catching up with computation using another machine holding a copy of the data in question. However, less sophisticated approaches might require manual steps from a calling application's side [Ell14, p. 25].

Scalability. Systems that can scale with increasing problem sizes are typically *horizontal scaling* systems (cf. Section 2.2.1). Especially for streaming systems, it is crucial to minimize the amount of necessary coordination between several machines, e. g., by ensuring that distributed tasks are partitioned so that they are as independent from one another as possible [Ell14, p. 27]. For enhanced coordination needs, dedicated software is available (e. g., Apache ZooKeeper, see Section 2.2.5). Another feature falling into this realm is *auto-scaling*. Using this, newly added processing nodes are automatically recognized and start working. Without this, adding new nodes to a cluster might require manual changes to resource allocation (e. g., cf. [GDK18]). *Vertical scaling* is also possible, but overall less effective for scaling.

Fault-tolerance. When there is a failure in the collection part of a streaming system, it must be decided through which mechanisms operations are resumed so that messages can be replayed (see *delivery guarantees* for details on specific behaviors from a consumer side). Notably, several collection nodes still maintain general availability while another collection node is down. Psaltis sees logging as primary feature for recovery in case of faults. In contrast, taking snapshots of a global state of a stream is not feasible and is better used with HDFS and other forms of persistent storage [Psa17, pp. 28ff.]. Logging can be done at once after receiving a message, accepting negative impacts on throughput, or before messages are relayed to data flow management. Hybrid approaches exist as well [Psa17, pp. 31ff.]. Similar considerations apply to data flow management [Psa17, pp. 52ff.]. After data is handed over to a Stream Processing Engine, fault-tolerance needs to be ensured as well, for instance through recovery by rollback or state-machines [Psa17, pp. 75f.]. Notably, different tools may offer varying degrees or implementations of fault-tolerance.

Latency. Streaming systems need to consider different types of latencies (as mentioned above) to minimize them according to their system requirements. For instance, for data latency, a collection system handling incoming streaming data must be able to handle a sufficient amount of incoming connections to bear the load of streaming data. Otherwise, the number of servers

must be increased [Ell14, p. 26]. Regarding analysis latency, needs for synchronization (i. e., determining whether a consumer has actually received a package) must be weighed against latencies induced by it [Ell14, p. 26]. If loss of messages or data to a certain degree is acceptable, latency can be further decreased [Ell14, p. 26]. In general, this is also dependent on, e. g., the type of *windowing* and *delivery guarantees* needed.

Windowing. To perform computations — especially aggregations — in an infinite and large stream of data, the relevant portion of data for these must be selected [Psa17, pp. 80f.]. This portion or excerpt is called *window*. Naturally, windowing influences latency as it dictates when and how many data items are passed to processing. Techniques for it rely on *trigger* and *eviction* policies [Psa17, p. 81]. The former dictates when processing code is notified to process all data items in a window, while the latter defines under which circumstances a data item is evicted (i. e., removed) from a window. Two basic windowing techniques are:

> **Sliding Window.** This technique uses time to define a window length (eviction policy) and sliding interval (trigger policy). Assuming a sliding interval of one second and a window length of two minutes means that processing code is notified each second to process all data, either in the sliding interval or in the whole window of two minutes. Data older than two minutes is evicted and not considered further. This could allow to, e. g., calculating an average value over the last two minutes, updating it each second. As time move forwards, so does the sliding window.
>
> **Tumbling Window.** Tumbling windows rely on the concept of either passage of time or counting data items in a window to decide when to trigger and evict. A stream is full and evicted completely, when either a set amount of time has passed, e. g., two seconds (temporal-based tumbling), or a sufficient number of data points are in the window (count-based tumbling) [Psa17, p. 83]. The latter works irrespective of time. If processing should only be triggered when there are at least 1000 items in the stream then this can happen after a few seconds, after

several hours, or even later than that. This is useful when, e. g., data comes in irregularly and a reaction to a set amount of data points should follow (cf. [Psa17, p. 84]). A notable difference between temporal-based tumbling and sliding windows is that the former evicts all collected events at once and clears the window until the next trigger. A sliding window with the same values for window length and interval equals a temporal-based tumbling window.

Sliding windows and temporal-based tumbling windows can be generalized as *time-based* techniques, whereas *count-based* technique is considered another category. Notably, support for *event time* vs. *stream time* by an SPE is crucial for windowing, as it influences aggregate computation. A more detailed view on different types of windows is provided by [CM12].

Delivery Guarantees. Processing high-velocity data in distributed stream processing systems poses a challenge when dealing with service disruptions in one or more worker nodes, i. e., partitions faults. When processing is interrupted because of this, messages from the stream are excluded from processing. If and how to recover from this is an important decision. Stream processing systems offer varying degrees of *delivery guarantees* [Psa17, p. 47]. On the one hand, high-value operations such as bank transactions require that no message is lost or duplicated due to recovery. On the other hand, approximate or iterative algorithms are less influenced by occasionally lost messages, as anomalies are likely to be averaged out in the long run (cf. [Psa17, pp. 48ff.]).

These delivery guarantees can be categorized into three distinct dimensions [Nas16]:

At-Most-Once. The application guarantees that each message is processed once at the most and never more often than that. This means that messages lost due to partition faults are not recovered and are effectively discarded. Conclusively, no duplicate messages occur. This is the loosest and simplest guarantee, as no specific mechanism needs

to be implemented. It is the default behavior when a system does not attempt any form of recovery. In this scenario, applications need to accept that they may not be able to process all messages, but do not need to deal with the occurrence of duplicates. [Nas16, p. 3]

At-Least-Once. The application guarantees that each message is processed at least one time. This means that no message is lost for processing, but that, e. g., during recovery, duplicate message may be created and are processed. An *at-least-once* implementation requires tracking of message before they are sent to distributed worker node. If a failure or timeout occurs, the message is relayed again to the recovered or another distributed worker node. If the latter has processed the message prior to failure or shortly after the defined time-out period, the message is processed twice. [Nas16, p. 3]

Exactly-Once. The application guarantees that each message is processed precisely once, without duplicates or discarded messages. This is the strongest and most reliable form of guarantee that can be offered. This makes it is useful in applications that need the most accurate data, e. g., banking transactions or fraud checks. Nassir et al. mention a possible implementation using checkpoints for each message. With these it can be decided if a message must really be relayed again after a failure. A checkpoint in a later stage ensures that processed messages are appropriately marked as such so that this information is available for failure recovery [Nas16, p. 3].

Out-of-order Processing. Usually, distributed systems in the most recent generation do not provide ordering semantics (e. g., Apache Kafka or Flume, see below). This means that messages are not necessarily delivered to data consumers in the same order as they have been produced, respectively generated. However, this allows to exploit distributed processing, where no further synchronization between nodes is necessary [Ell14, p. 19]. Synchronization may introduce sequential program parts, which can hinder parallel performance increase (e. g., cf. [Amd67]). Dedicated message queuing platforms

such as Apache ActiveMQ[50], labeled as predecessors for aforementioned modern systems, made dedicated efforts to offer in-order-processing. Ellis postulates that this task is more effectively handled by data consumers [Ell14, p. 19].

Technology Overview

For each part of a streaming architecture, several technologies (i. e., SPTs) have emerged. These include platforms, frameworks, processing engines, dedicated tools, and libraries. A selection of notable technologies directly related to streaming is briefly described in the following. The goal is to illustrate the present stream processing technology landscape through notable representatives and outline their properties with respect to the previously introduced technology characteristics. Several of these are projects under the umbrella of the Apache foundation and available free of charge, including their source code. A comparative overview of these is depicted in Table 2.6. Generic technologies from different fields, such as NoSQL data stores, are not further detailed here, although they might be used for storage and other supporting aspects.

Apache Storm [13]. Storm is a distributed, native stream processing engine. Its graph-based composition topology represents computations starting from input streams (*spouts*) to various stages of output streams (*bolts*). As is typical for stream processing, these deployed topologies do not simply end like MapReduce jobs, but can keep being executed [CZ14, pp. 328f.]. Its concept of a master and several worker nodes connected to streaming sources (named *Nimbus* and *Supervisor*) [Psa17, pp. 66f.] resembles job and resource management in Hadoop clusters [CZ14, p. 329], whereas jobs as known from MapReduce are represented by an executable topology here [Psa17, p. 67]. Storm's distributed architecture allows for horizontal scaling by adding more nodes, but necessitates manual adjustments to do so efficiently (cf. [GDK18]). As alternative interface to Storm, Trident [102]

[50] http://activemq.apache.org.

provides built-in aggregations, support for micro-batches, and allows for exactly-once delivery guarantees at the expense of higher latencies.

Apache Spark [76] and Spark Streaming [98]. Spark in general is a distributed execution engine relying on DAGs and in-memory computing. This approach is comparable to MapReduce and other related components (e. g., MapReduce uses Apache Hive for querying of its data similarly as Spark uses SparkSQL). Spark seen as part of the Hadoop ecosystem, but can be run independently as well (cf. [Sto14, ZCD+12] [Psa17, p. 66]). In contrast to MapReduce, it aims to unify batch, interactive, and streaming analysis into one platform. Its key concept for data abstraction are so-called Resilient Distributed Datasets (RDDs) [ZCD+12, pp. 157f.], which are immutable, read-only sets of data kept in main memory or secondary storage, if the former is full. Each operation on an RDD creates a new RDD. This allows to restart computations at a specific RDD, for example, after failures. With MapReduce, usually a job must be restarted from scratch when failures occur [ZCD+12, pp. 159]. Spark Streaming is built on top of the general-purpose engine Spark and enables the latter to analyze streams of data using RDDs. This is done by summarizing streaming data into micro-batches, so-called discretized streams represented by RDDs, and relays these to the Spark engine for execution [ZDL+13, pp. 425f.]. This implies that data is processed in-order at least per discretized stream, but apart from that order is not guaranteed [97]. Notably, Spark 2.0 introduced *Dataset* and *Dataframes* as unified abstractions with a new API for both batch and streaming use cases, which are built on-top of RDDs and are alternatives to the previously mentioned APIs [153]. Due to the nature of micro-batch processing, latencies are limited in comparison to native stream models. The goal of this overall approach is to improve fault-tolerance and provide exactly-once delivery semantics [Sto14]. In addition to a standalone mode, YARN and Apache Mesos can be used for cluster management [81]. Besides Streaming, SparkSQL as query language and MLlib [91] for Machine Learning (see Section 2.3) are supported [76] [Psa17, p. 65].

Apache Flink [20]. Similarly to Apache Storm, Flink is a distributed native stream processing engine, leveraging a dataflow-based streaming model [CEH+15, pp. 30f.]. Flink's approach is comparable to Apache Spark as Flink attempts to unify both streaming and batch analysis with distinct operator APIs (DataStream and DataSet API, cf. [19]). Additionally, there is a Table API[51] allowing table-based access to data flows with an SQL-like query language [58], with a syntax based on Apache Calcite [46]. In addition to that Flink can simulate micro-batches, which allows for stateful aggregate operators and batch processing. Moreover, exactly-only delivery guarantees can be supported. Fault-tolerance is further implemented using "lightweight" checkpoints [52] [CEH+15, pp. 31f.; p. 37]. Notably, it is assumed that data sources are fault tolerant as well and can replay data after a failure for partial re-execution. Flink supports sliding and tumbling windows (count- and temporal-based), while also allowing event time to be considered instead of only stream time to avoid time skews [CEH+15, pp. 33f.]. Flink can use YARN and Apache Mesos for resource and cluster management or be running in a standalone mode. Apart from that, Flink has a dedicated machine learning library, FlinkML [CEH+15, p. 30] [55] as well as connections as sink or source to other related projects such as Apache Kafka or HDFS [18].

Apache Kafka [23] and Kafka Streams [27]. Apache Kafka is a distributed publish-subscribe messaging system, built on the concept of a "distributed commit log" [WKS+15, p. 1654]. It fulfills the role of collection and data flow management in a typical streaming architecture (see above). Consequently, it offers separation and message queuing between data producers and consumers connected by the concept of *topics* [25]. Generally, Kafka does not handle out-of-order messages unless they are contained within a smaller management unit of topics called *partition* (cf. [25] [Ell14, p. 19] [KK15, p. 9]), which enables horizontal scaling [KK15, pp. 3ff.]. Kafka stores data on disk and replicates it to provide fault-tolerance [25], relying on the efficiency of sequentialized disk-access and exploiting certain OS mechanisms [24]. By default, Kafka guarantees at-least-once delivery. Reliable exactly-once delivery is possible, but relies on coordination with a connected processing

[51] As of Flink 1.4, this API is marked as "beta" and is still in active development [58].

engine [24]. Kafka Streams is a library and native component of Kafka since version 0.10 [15] and enriches it with generic stream processing capabilities. It is designed as a "lightweight" library, leveraging Kafka's capabilities for fault-tolerance [26]. Coordinating with Kafka, it supports "end-to-end" exactly-once delivery as well as windowing using (event) time [26]. Due to its reliance on Kafka, Kafka Streams may need manual processing to completely guarantee ordering semantics besides order guarantees outside a partition.

Apache Flume [40]. Flume is a distributed system for *log* data collection and data flow management includes some aggregation capabilities. While Flume is designed to do so for Web log data, collected from various sources and pushed to a central target location [40], other data sources can be used as well. The authors explicitly mention e-mail messages or network traffic data [31]. Its key concept is a channel through which data moves from source to sink. Besides HDFS and Kafka, several enterprise and legacy sources, like syslog files, can be processed with Flume as well [137]. Using Kafka is the advised way to integrate Flume with some of the aforementioned tools like Storm [Ell14, p. 141]. Ellis recommends Flume to store streaming data in persistent storage and moving other kinds of "legacy" data [Ell14, p. 115]. Flume supports horizontal scalability and uses Apache ZooKeeper for coordination. For reliability, an acknowledgment by the sink is requested and, if not received, data is retransmitted [31]. Further, the order of messages cannot be guaranteed by Flume [Hof15, p. 242] and at-least-once deliveries are ensured [9]. Multiple data flows in separate agents can be combined for, e. g., integration tasks [Hof15, p. 54]. As no sophisticated processing capabilities are offered, there is no support for windowing. Recoverability is provided on a channel-level and there for disk-persisted data only [31].

Apache Samza [95]. Apache Samza takes the role of a stream processor, where user-implemented code is executed on streaming data. Kleppmann mentions it explicitly as complement to Apache Kafka [KK15, p. 3]. It consumes data from Apache Kafka *topics* organized in *partitions* on which a Samza StreamTask is executed [KK15, p. 5]. Afterwards, it sends processed data back to a Kafka output [KK15, p. 5]. This is similar to MapReduce jobs

being executed using HDFS [KK15, p. 8]. In contrast to Kafka Streams, Samza jobs are managed by Hadoop YARN [KK15, p. 5]. For persistence to recover from failures, Samza uses RocksDB[52] or any other key-value store implementation provided by users that is run on the same machine as the execution task. As any message from a previous checkpoint until a failure is replaced, at-least-once delivery can be guaranteed [KK15, p. 9]. Ordering, as in Kafka, can be ensured on partition level [KK15, p. 9].

Apache Apex [42]. Apex unifies distributed batch and native stream processing in one platform while relying on YARN ("YARN-native" [42]) for management and HDFS for persistence and fault-tolerance [42, 225]. Thus, it is horizontally scalable. It supports windowing with event times and utilizes checkpoints for recovery instead of acknowledgments (cf. [79]). To connect to data sources, any source compatible with Java is suitable, which includes HDFS, DBMSs, NoSQL data stores, files, and other processing engines such as Kafka [225, 44]. Operators and functions are separated. The goal is to decouple business logic to allow for either batch or streaming execution of code [225]. However, not all operators are considered equally mature as Weise et al. note [WRYK17, p. 78], because several operators mentioned in the documentation are not yet integrated in the main library folder on GitHub [17] (cf. [399]). Notably, its included operator library, Malhar, inherently supports data analytics in form of, e. g., statistics and selected algorithms[53]. In addition to that, custom user code written in languages such as Python or R can be invoked [44]. Furthermore, using Apache SAMOA[54] [132] with Apex is suggested for machine learning and analysis (cf. [MB15]).

Further Streaming Processing Technologies. *Twitter Heron* [KBF+15] means to improve upon Apache Storm regarding the scalability and manageability of Storm topologies while remaining API-compatible to the latter.

[52] http://rocksdb.org.

[53] While "Machine Learning" and "Pattern Recognition" are listed as Malhar features, they are neither explicitly documented nor is there readily usable code for this in the master branch of the library on GitHub (cf. [43]).

[54] SAMOA is discussed separately in Section 4.2.2 and Section C.1.4. It is also compatible with, e. g., Storm (cf. [10]).

Table 2.6: Properties of various Streaming Processing Technologies and their characteristics. Based on: [Psa17, p. 82; p. 85], [GDK18], [73], [74], [286], [CEH+15], [KK15], [168] [96] [WRYK17] [79].

	Storm	Spark Streaming	Flink	Kafka Streams	Flume	Samza	Apex
Streaming Model	Native	Micro-batch	Native	Native	Native	Native	Native
Scalability	Horizontal	Horizontal (Auto)	Horizontal	Horizontal (Auto)	Horizontal	Horizontal	Horizontal (Auto)
Windowing	Tumbling (Temporal, Count)	Sliding, Tumbling (Temporal)	Sliding, Tumbling (Temporal, Count)	Sliding, Tumbling (Temporal)	✖	Tumbling (Temporal)	Sliding, Tumbling (Temporal)
Delivery Guarantees	At-least-once, Exactly-once[iv]	Exactly-once, At-least-once[v]	Exactly-once	At-least-once, Exactly-once	At-least-once	At-least-once	At-least-once, At-most-once[vi], Exactly-once[vii]
Out-of-order Processing	✔	(✔)	✔	(✔)	✖	(✔)	✖
Fault-tolerance	✔	✔	✔	✔	✔	✔	✔

[iv] With Trident. [v] If source is not fault-tolerant [224]. [vi] E. g., using the Malhar library [44]. [vii] During Windowing [398].

Apache Gearpump [21], a currently incubating Apache project, is a distributed stream processing engine that aims towards real-time environments and high throughput with up to exactly-once delivery guarantees. It is based on an event/message interaction pattern and builds on in-memory storage [82]. *Apache S4* [12] is a retired Apache project, which also was distributed stream processing engine, discussed in various publications (e. g., cf. [LLD16, Nas16, MB15]). *Google Millwheel* [ABB+13] is a private, distributed stream processing platform by Google, focusing on fault-tolerance and scalability and offers exactly-once guarantees. *Apache NiFi* [29] is a data flow management software designed to move various data sources and can be configured for high-throughput or low-latency scenarios, which include streaming data cases (cf. [395]). *Apache Ignite* [35] is an in-memory platform for both storage and processing aimed towards streaming workflows and is designated to provide SQL and transactional semantics (i. e., transaction safety) while allowing horizontal scalability. *Apache Beam* [45] offers a pipeline model which can be executed on several of the aforementioned stream processing engines such as Apache Flink, Spark, Apex, or Gearpump as well as MapReduce [80] (cf. Section 2.2.4).

2.2.7 Advancements to Traditional Technologies

Since their inception, traditional technologies, namely RDBMSs and Data Warehouses, underwent several advancements, which include invigoration of existing ideas as well as new approaches in the light of NoSQL data stores. These advancements are IMDBs, databases belonging to the so-called *NewSQL* category, as well as modern Data Warehouses. The first two are presented in the following. Modern Warehouses are separately discussed in Section 3.3.

In-memory Databases. The difference between IMDBs and regular DBMSs is that IMDBs have (volatile) main-memory as primary storage location and secondary storage (e. g., hard-disks or SSDs) only for persistence and backup purposes. Main advantages of main memory are both access latency in the nanosecond range as well several orders of magnitudes higher throughput

(cf. [Jac09]). Disadvantages are related to limited size of main memory in comparison to secondary storage, although serves with several TBs of main memory are already available (e. g., [108]). Open challenges, respectively trade-offs remain regarding persistence mechanisms due to memory volatility and its effects on transaction execution (cf. [GMS92, pp. 511f.]). RDBMSs store data mainly on secondary storage, while keeping only caches or index structures in main-memory for quick lookups[55]. Another difference is that query execution and internal organization is adapted to specifics of main memory (cf. [PA16, p. 47]). Otherwise, one could simply use a virtual drive partition mapped to main memory ("RAM disk") as storage location for a relational database and label it IMDB. However, a regular relational row-based query executor is typically tuned for slower secondary storage [PA16, p. 47]. Especially due to hard disk latency, these try to sequentially access patterns to take advantage of a hard disk's optimal performance capabilities, because random access on disks is several orders of magnitudes slower (cf. [Jac09]). Proposals for main memory databases were already made decades ago in the 1990s (cf. [GMS92]). As indicated in Section 2.2.1 they only became a viable option in the last decade as both capacities and price of RAM reached feasible levels (cf. [GH03, p. 341] [254]). The IMDB category is an orthogonal category, meaning that both relational as well as NoSQL data stores offer in-memory storage. For different needs on data variety, different systems can be chosen. *Redis* as in-memory key-value store is presented in Section 2.2.2. Further, SAP HANA as in-memory Data Warehouse is discussed in Section 3.3 and in-memory RDBMS are illustrated below. Generally speaking, IMDBs are best suited to tackle velocity challenges. In that sense, they can process huge data volumes quickly as well, but they are less fit for persistent storage of large data volumes than disk-based storage systems, even in distributed scenarios.

NewSQL DBMSs. NewSQL systems are related to the NoSQL movement. However, their goal is to provide relational data structures including SQL as query language and consistency (ACID guarantees), while still maintain scalability for typical application workloads, e. g. OLTP, as offered by

[55] Some RDBMSs also offer to store whole tables temporarily in-memory, e. g., MySQL [303].

Table 2.7: List of NewSQL systems examples with primary form of storage (disk or in-memory) and default focus regarding CAP properties.

System	Storage	CAP Focus
Google Spanner [CDE+13]	Disk	CP
Google F1 [SEC+13]	Disk	CP
H-Store[56]	Disk	CP
CockroachDB [140]	Disk	CP
VoltDB[57]	Disk	CP
S-Store [CTT+14]	Disk	CP
MySQL NDB Cluster [301]	In-memory	CP

other NoSQL data stores [PA16, p. 46] [369]. These systems also feature a distributed, shared-nothing architecture and a query executor, which can exploit this [PA16, p. 47]. Pavlo and Aslett note that organizations may find these tools challenging to use, as they are relatively new and may lack maturity and tool support (e. g., for monitoring and administration) known from established, traditional RDBMSs. Notable exceptions attempt to maintain protocol compatibility with established tools [PA16, p. 46]. Several of these new systems use decentralized Multiversion Concurrency Control (MVCC) for concurrency control [PA16, p. 50], which works optimistically (instead of a pessimistic approach for conflict avoidance like 2PL). It allows for higher read and write throughput, which is particular important for scalability in distributed storage systems, while accepting a different behavior for concurrency-related anomalies (cf. [EN16, pp. 795ff.]). Newer implementations of MVCC could even provide full serializability of transactions instead of focusing on snapshot isolation (e. g., [NMK15]). Importantly, prominent exemplars of NewSQL DBMSs feature a relational model at a logical level, but rely on key-value based technology on their lowest layers, such as Google F1 [SEC+13] or Cockroach DB [142]. Nevertheless, they are still categorized as distributed RDBMSs due their reliance on a logical relational model with consistency guarantees as well as SQL. A list of NewSQL examples is listed in Table 2.7.

2.3 Big Data Analytics

Big Data Analytics is one of the key aspects to extract value out of Big Data. After outlining how storage and processing technology can address Big Data characteristics in the previous section, the focus is now on analytics. This section presents an overview of approaches to analytics with special consideration to new developments fueled by Big Data. First, aspects of modern and traditional analytics are outlined in Section 2.3.1. Afterwards, in Section 2.3.2, the new "hype" topic of machine learning is elaborated upon, before typical analytics methods are presented and described in Section 2.3.3. Lastly, new application domains of analytics due to Big Data are then discussed in Section 2.3.4.

2.3.1 Analytics in the Big Data Age

Data analytics can be classified into three distinct types: *Descriptive Analytics*, *Predictive Analytics*, and *Prescriptive Analytics* [IBM13, SZS15, vR14]. These categories try to answer a different set of questions and have each a certain set of analysis techniques associated with them.

Descriptive Analytics. Like descriptive statistics, the intent is to identify patterns in data ex post, i. e., on data which refers to the past. In that sense, it "describes" a fact about the data by means of analysis. This is done in hindsight to expose insights, or facts, about events that already have happened ("What has happened?" [IBM13, p. 3]). Typical DWH applications fall into the category of descriptive analytics. Business reports, KPI dashboards, and OLAP cubes allow to see and inspect, e. g., sales data from transactions that were already completed prior analysis. Thus, descriptive analytics is the most traditional form of analytics and, also, the most common form of analytics [IBM13, p. 3]. Within the field of retrospective analyses, some authors also see *diagnostic analysis* distinctly. Its goal is to find a root cause to the phenomena identified by other descriptive statistics. However, it still uses descriptive methods to search for such causal links, e. g., drill-down- and roll-up-based analyses [253].

Predictive Analytics. It has the goal to foretell ("predict") future events or behaviors based on data. While data from the past is indeed used, the focus is to forecast what might be the case in the future ("What could happen?" [IBM13, p. 3]). As Han et al. describe, predictive tasks use induction on present data to calculate predictions [HKP11, p. 15]. A typical example is a sales forecast, where, based on past sales and sales patterns, future sales are predicted. By "anticipating [these] likely scenarios" [IBM13, p. 3], this can not only be used to project future turnovers and revenues, but also to impact decisions such as needed warehouse capacities. In many ways, predictive analytics shares traits with data mining (cf. Section 2.1.2). Gartner postulates that predictive analytics is an improved approach to traditional data mining associated with traditional BI. Specifically, they see a new set of techniques deliberately tuned towards prediction instead of focusing on descriptive methods for, e. g., classification and clustering. Moreover, modern predictive analytics efforts should not take months (as traditional data mining supposedly took), but rather hours or days. Lastly, the results should not be limited to mere insight, but also on business value and its promotion to business users through a more intuitive tool set [175]. A typical example of predictive analytics are weather forecasts. For instance, the National Weather Service (NWS) in the US employs supercomputers with up to 6 petaflops (billion floating point operation per second) with sophisticated prediction models using atmospheric, oceanic, and satellite-based information [390]. More modern forecasting methods try to correlate weather data with business information, e. g., needing to buy an umbrella when it rains [326]. Weather information can also be used when forecasting logistics, especially when shipping times are rather long and require planning ahead, e. g., when snow is expected, winter clothes can be moved from a warehouse to a store right in time [238].

Prescriptive Analytics. With this form of analytics, the concept of predictive analytics is further expanded by actually using predicted values to steer and optimize an organization as well as its decision making. In this case, predictions are turned into a code of conduct, i. e., they prescribe what to do. In that sense, they can give more specific recommendation on which

base actions can be taken ("What should be done?" [177]), sometimes even in a fully automated fashion. At core, methods of predictive analytics are used. Wood Squier from blog datasciencecentral.com describes prescriptive analysis as ideal towards which *good* analysts should strive for, as mere predictions without actionability and context would be of limited use [401]. One example for prescriptive methods are automated pricing mechanisms in the retail industry. Recent trends include to adapt the price to the properties of the buying customer to better match their willingness to pay the to-be-asked price. Other uses are monitoring competitors and adjusting prices automatically [1]. For instance, in 2006, Hwang and Kim described mechanisms for automatic price adjustments based on algorithms [HK06]. Another area that received widespread attention is autonomous driving. The ultimate goal is that a car is able to navigate without human interference, using several sensors and cameras, on the road and can "intelligently" react to any kind of unexpected situation, e. g., a child running in front of the car or blocked roads. Several types of methods are needed to solve the surrounding issues [342].

Complexity of analyses is attributed to be increasing from descriptive to prescriptive analyses. Franks also notes that organizations must adapt their processes and culture to leverage these sophisticated forms of analytics [Fra12, pp. 253f.].

The term **Advanced Analytics** is often used to relate to novel Big Data analytics. It is especially distinguished from so-called "BI analytics" [Rap14, p. 2], which, however, refers mostly to *traditional* BI analytics (cf. Section 2.1) [174]. The latter is used to summarize several descriptive BI methods, in particular OLAP, dashboards, reports, and ad-hoc queries. Advanced analytics is used to describe any approaches to solve challenges and find insights with "sophisticated" methods [174]. Its main contributions come from the fields of *data mining* and *machine learning* [Rap14, p. 2]. These insights from advanced methods were not available in traditional BI analytics [174]. Franks asserts that "core analytics", which he relates to reporting, is defined by "simple" questions and answers through descriptive analyses. In contrast to that, he sees advanced analytics as encompassing predictive and prescriptive analyses, but also a part of descriptive methods, in particular those

aimed at identified root causes (diagnostic analytics) such as "highly complex SQL queries", which combine data sources in "complex ways" [Fra12, pp. 186f.]. Apart from that, he sees predictive modeling, data mining, forecasting, optimization, and "similar activities" as typical activities [Fra12, pp. 187]. Gartner lists the following techniques as typical examples of advanced analytics: "data/text mining, machine learning, pattern matching, forecasting, visualization, semantic analysis, sentiment analysis, network and cluster analysis, multivariate statistics, graph analysis, simulation, complex event processing, neural networks" [174].

However, strictly speaking, both traditional and modern BI analytics are not limited to descriptive and other traditional methods. For instance, data mining is a BI-related tool, which can also be used for predictive analyses. Gartners BI capability matrix from 2007 also sees predictive methods as part of needed BI capabilities [SS07]. Especially modern forms of BI analytics need to incorporate advanced analytics, which includes predictive methods. In a literature study published in MIS Quarterly [CCS12], Chen et al. conclude that three evolutionary phases of "BI and Analytics" (BI&A) exist. They see traditional BI as first iteration *BI&A 1.0* and relate most of the aforementioned Gartner BI capabilities to this phase [CCS12, p. 1166], which focuses on structured content. With *BI&A 2.0* they see a shifted focus towards adding Web 2.0 analyses of semi- and unstructured content (e. g., analyzing Web logs or social media, and other Web analytics) [CCS12, p. 1167]. As of 2012, the third phase, *BI&A 3.0* was still emerging. It would be characterized by analysis of data sourced from many "new" mobile devices and IoT devices such as sensors ("Web 3.0"). However, due to the infancy of these topics, not all analytics methods for this would be known yet and integrated commercial systems were yet to be introduced [CCS12, p. 1168]. According to this study and Gartner, modern BI must be capable to deal with various kinds of business data beyond structured tabular data and offer rapid access to analyses besides classical reporting and dashboards (e. g., via smartphone as so-called "mobile BI"). Other recent trends for modern BI include self-service processes for added flexibility for users, who can add sources and easily specify their transformation themselves by means

of intuitive software ("self-service BI"). These trends have gained severe tractions since the paper by Chen et al. in 2012, especially through the even more widespread use of smartphones and advancement to the area of IoT (cf. [176, 179, TS16]).

To gain a better understanding of the various types of available analytics (e. g., the ones mentioned by Gartner), the specified techniques employed therein need to be elaborated upon and structured, which is done in the following sections. Next, the field of *machine learning* is described alongside the latest trends surrounding methods in the field of AI, which are often related to machine learning.

2.3.2 Machine Learning

Machine learning in general deals with a computer's "learning" of existing knowledge by example (data). Through learning, this knowledge is codified and can be applied, preferably automatically, to existing or new data [HKP11, p. 24]. This codification is conducted by building statistical models [Rap14, p. 3]. From the perspective of human teachers, a model is trained with sample data to extract and describe those structural patterns, which enable explanation and prediction [WFHP16, pp. 2ff.]. Generally, a statistical model is a "set of mathematical functions" [HKP11, p. 23] characterizing data item behavior, both in respect to random variables as well as their probability distributions [HKP11, p. 23]. The associated terms with descriptive and predictive analytics are thus descriptive and predictive modeling (e. g., cf. [Rap14]).

While related to machine learning, data mining (cf. Section 2.1.2) generally refers to generating "new" knowledge and rules. Further, its meaning expands to the overall process and not only the aforementioned learning. However, machine learning is a newer term and is widely used nowadays. In fact, it is currently regraded a "hyped" emergent technology by Gartner [187]. Notably, machine learning techniques can be used for data mining as well (cf. [HKP11, p. 23]), just as data mining can be used, besides machine learning methods, for predictive analytics [Rap14, p. 3].

Machine learning techniques can be categorized into three types of learning:

Supervised Learning describes the learning of patterns by appropriately labeled example data. For instance, a training data set could contain various customer master data and an attribute indicating if the customer has canceled his contract, i. e. there a binary label to each customer. Using this, a model can be trained to predict if a yet unknown customer is likely to cancel or not. A typical application for this is classification (see next subsection). [HKP11, p. 24]

Unsupervised Learning uses, in contrast to the aforementioned, unlabeled data. That means that only the given data is used to determine patterns. Cluster analysis (see next subsection) is one example for unsupervised learning, where a similarity function determines grouping (clusters) of data. [HKP11, p. 25]

Semi-supervised Learning is a mixture of both supervised and unsupervised learning, where both labeled and unlabeled training data is used. Han et al. describe that labeled data is used for basic learning, and unlabeled data is then used to refine the initial results. [HKP11, p. 25]

Han et al. further describe *active learning* as an alternative form, where human users actively participate in the learning process and help to improve model quality through manual adjustments [HKP11, p. 25].

Several Big Data application of machine learning can be found in practice. For instance, Google engineer Kaz Sato used Google AutoML Vision[58], a cloud service for machine learning, to train a model with food imagery to identify (classify) its restaurant of origin by the way it was prepared by their various chefs (e. g., chipped meat size) [358]. Interestingly, in "base mode" a model can be created in 18 minutes, while a more precise "advanced model" takes 24 hours to compute. Such trade-off models are commonly called *commodity models*, where a fit between time-to-compute and result quality is considered [Fra12, pp. 157f.]. In this example, the 18-minute basic model was already deemed as reasonably accurate. Other applications include healthcare (e. g., automatically evaluating MRI images), autonomous vehicles (e. g., detecting objects in visual inputs), online search (e. g., making images

[58] https://cloud.google.com/automl/.

searchable), or financial trading (e. g., predicting future stock prices for automated trading) [248].

Artificial Intelligence, Neural Networks, and Deep Learning

The area of AI has become a widely popular topic in the field of analytics in recent years (cf. [187]). As a term, AI has several facets. Russell and Norvig see acting and thinking either humanely or rationally as aspects of AI [RN09]. The human aspect aims to replicate specific capabilities that are associated with the human mind [RN09, p. 2]. Thinking and acting rationally on the other hand focuses on rational and intelligent agents, which strive for the best possible outcome by making "correct" inferences [RN09, p. 4]. Notably, achieving the best possible outcome in all conceivable contingencies is often too computationally intensive, requiring certain trade-offs [RN09, p. 5]. For the field of machine learning, Russell and Norvig postulate that an AI in general needs to "adapt to new circumstances and to detect and extrapolate patterns" [RN09, p. 2].

Particularly *neural networks* and its subfield *deep learning* are often associated with AI, although several other methods and techniques were developed and used in this field (cf. [RN09]). (Artificial) neural networks are modeled after the human brain, which possesses a system of interconnected neurons capable of reorganizing its connections (e. g., cf. [PCN+11]). In an (artificial) neural network, a system of processors called neurons calculates outputs according to respective input values. Different inputs lead to different neurons being activated and, consequently, varying outputs. When repeating this, a neural network adjusts the weights (i. e., influence) of its neurons so that a desired output is reached eventually. Using this principle, a neural network can learn through self-adaptation. For instance, by supervised learning, a neural network adapts its weights to match inputs to the desired outputs through its network of neurons. A critical aspect are layers and the number of neurons per layer, which are chosen beforehand[59]. Computationally

[59] In the human brain, reorganization happens by changing the connections between neurons. Unless neurons die, their number remains constant, as in most artificial neural networks, although there are newer dynamic approaches as well (e. g., [SD14]).

intensive problems might require not only a few, but many stages. In case of many stages, the term *deep learning* is used, as a high number of neuron layers renders the overall network "deep" [Sch15, p. 86]. Usually, only the input and outputs are visible to a user of a neural networks. Neurons inside a network are also called "hidden inputs" as they quietly influence the outcome [LWL+17]. Schmidhuber summarizes both historical as well as recent methods for (deep) neural networks [Sch15]. While neural networks help to avoid manual tuning of parameters, the created networks are less intuitive to comprehend by human users ("black box") and the interpretation of their result may be challenging (cf. [TS92, p. 977]), although several techniques for doing so start to emerge (e. g., cf. [TS92, MSM18]). A proper design and evaluation is also crucial, as some simply trained neural networks may behave unexpectedly, e. g., being stuck in local optimum instead of finding a globally optimal solution or over-fitting the network to the given problem, hindering generalizability towards new data [LWL+17, pp. 11f.].

Applications of neural networks are manifold. Especially the aforementioned examples for predictive analyses rely on AI and neural network-based approaches. For instance, Grzywaczewski, an engineer for company nVidia[60], outlines challenges for training AIs in autonomous cars, especially regarding required car fleet size for meaningful data collection and computational power needed to actually train neural network models [193]. Further, Google[61] uses neural networks, for image recognition for their search application, where classified images should allow users to search for certain criteria of these. Apart from this, they use it to enhance images by adding details missing or distorted by considering other images (e. g., when several pictures of the Brandenburg Gate are known to a neural network, it could use this information to automatically complete a partial image of the gate) [249].

[60] https://www.nvidia.com.
[61] http://www.google.com.

2.3.3 Methods

Based on the typical methods of data mining as categorized by [HKP11] (see also Section 2.1.2) and for predictive analytics by [LL15], in consideration of typical DWH and OLAP functionalities (cf. Section 2.1.2) as well recent additions through Big Data analytics and machine learning, the following categories of data analytics functionalities, covering both traditional BI analytics as well advanced analytics, can be identified:

Data Characterization and Discrimination. These two descriptive activities have the goal to unearth general features and characteristics of the data at hand. *Characterization* is about *summarizing* general data features [HKP11, p. 15], while *discrimination* is about comparing general features to other data items [HKP11, p. 16]. According to Han et al., one of the main use cases for these is to define or set class labels for certain classes (i. e., groups) of data by defining the features of these classes (cf. data classification below). Several statistical descriptors are generally suitable for these tasks. For instance, there *measures of central tendency*, namely mode, mean, or median [HKP11, pp. 39ff.], as well as *measures of dispersion* such as range (difference of maximum and minimum value), quartiles, variances, or the standard deviation [HKP11, pp. 48ff.]. These are summarizing features, i. e., aggregate measures over data (cf. [HKP11, p. 145]). Beside the aforementioned statistical descriptors, also simpler measures such as for counting data items in a group (`COUNT()`), summing up values (`SUM()`), or determining minimum or maximum values are suitable. A large set of OLAP functions cover this analytics category, often supported by SQL. Drill-down, roll-up, and other typical operators offer interactive means to calculate and compare these descriptors on different subsets of data [HKP11, pp. 153f.]. Possible outputs can be manifold. Besides simply outputting data as regular tables [WFHP16, p. 63], it can be visualized as charts or graphs (e. g., a bar chart or a box plot), result in an OLAP cube, or even in a set of rules (e. g., in "If-Then" format) [HKP11, p. 16]. A use case for these results can be to describe classes or groups of data or other functional concepts. For instance, an analysis of

the top 10 % of customers based on revenue could reveal that a certain age group is mostly responsible for these purchases.

Classification and Regression. A large set of different techniques and algorithms falls under these two categories. They are widely used for predictive analytics through a broad range of use cases.

> **Classification** methods and techniques partition data into distinct sets of data called classes, which are defined beforehand. Each of these is denoted with a predefined *class label*. Based on pre-existing knowledge about which data points (e. g., rows) belong into which class, the class membership of new data points can be predicted. This knowledge is expressed by attribution of each data point to a class and thus defining the distinctive attributes of these classes [Agg15, p. 18]. These class labels are discrete, unordered, categorical labels [HKP11, p. 19]. As expressed above, classification is a case of supervised learning [HKP11, p. 130]. For instance, in an online sales scenario, different customer archetypes can be used for marketing (e. g., urban, suburban or rural customer, when using geography as a classification feature). These classes can be associated by relating postal package destination information (to infer where the customers live) to sales transaction and, thus, to these customers buying habits. These classes could be used for targeted marketing initiatives, such as promoting fast deliveries for urban customers. A wide range of techniques is suitable for classification and can create a diverse set of output types. For instance, there are simple rule-based classifiers ("If-Then") [HKP11, pp. 355ff.] or decision trees with popular algorithms such as ID3, CART, or C4.5 [HKP11, pp. 332]. Other larger classes of techniques are Bayesian classifiers (e. g., [Agg15, pp. 306ff.]) and Support Vector Machines (SVM), which work by transforming data into a higher dimension [HKP11, p. 408ff.]. Further, there are k-nearest-neighbors [HKP11, pp. 415ff.], case-based reasoning [HKP11, pp. 425ff.], genetic algorithms [HKP11, pp. 426ff.], and semi-supervised methods [HKP11, pp. 432f.]. (Deep) neural networks represent another recently popularized form of learning for classification (e. g., cf. [HKP11,

pp. 398ff.]). Lastly, also outputs of other techniques can be used, e. g., in pattern-based matching methods, mined patterns and association rules are used to build a classification model. Another type of distinction is whether the methods are "eager" or "lazy" learners. In the latter case, learning can be easily incremental, as the learning process is not "eagerly" completed at once, but only conducted when prediction is demanded. This type of learning is also called instance-based learning. While there are higher storage requirements for "lazy learners", they can be efficiently run parallelized, which makes them well-suited for Big Data scenarios [HKP11, p. 423].

Regression is used instead of classification for "continuous-valued functions", i. e. numerical data, instead of categorical labels [HKP11, p. 19]. The field of statistics contributes several methods for regression, also widely used in mathematics. For instance, there are linear, multiple, weighted, or polynomial forms of regression [HKP11, p. 599]. Depending on the type of relationship between the data, a different type of technique can be used for estimation respectively prediction (e. g., in a linear case, an unknown next value is assumed to be one a straight line, which describes the generic trend of all data). Time-series analysis is one typical example for regression and used for, e. g., trend analyses and value forecasting. Here, data points over time such as sales transactions are the main input. Another example is forecasting revenues, where a mathematical function is found to describe the generic relationship between input parameters and output (e. g., a linear model such as $revenue(year, cost) = 2 * year + 250 - cost$) and allows to infer future values.

Cluster Analysis. It attempts to solve the issue of finding a statistical model that characterizes or discriminates data classes [HKP11, p. 18]. The general idea is similar to classification, but cluster analyzed works unsupervised. There are no existing class labels as input and the result of the cluster analysis are clusters (groups) of data that are not labeled. Thus, an interpretation of the result through further means should follow (e. g., by using data char-

acterization to find appropriate class labels). Clustering is not predictive, but descriptive. However, it can be used as foundation to build other predictive models using its output. A crucial point for cluster analysis is the concept of "similarity" and the respective similarity function, which ultimately defines the output clusters [Agg15, pp. 16f.]. There are several clustering methods available (cf. [HKP11, p. 448ff.]). Popular algorithms as k-means falls into the category of partitions methods. Moreover, there hierarchical ones (e. g., agglomerative clustering), density-based methods (e. g., DBSCAN), grid-based methods (e. g., STING) or methods based on a probabilistic model. A possible application scenario is customer segmentation, where similar customers are clustered so that they respond to certain marketing efforts (cf. [Agg15, p. 17]). A similarity measure here could be based on their previous sales transaction history (with KPIs involved such as number and product category of items per sales or overall sales volume). For text mining, a field interrelated with Natural Language Processing (NLP), e. g., *topic modeling* is one of the most popular methods [APA+17, p. 7].

Outlier Detection. This activity is tasked to find data points which differ severely from the rest of the data. Outliers can be suspected to represent an anomaly in the data generation, which renders its underlying mechanisms differently from the remaining data set. This brings up several challenges, but also opportunities when analyzing data with outliers. Whereas in some cases outliers should be removed from the analysis as they lead to distortions, the detection of outliers can reveal unwanted behavior in the source systems such as unexpected system faults. For instance, in computer logs, anomalies may point to suspicious behavior of malware or an intruder. Detecting such atypical data points is also the corner stone of credit card or online payment fraud detection. [Agg15, pp. 17f.] [HKP11, pp. 543ff.] Outliers can be either determined by inspecting a data point in relation to all other data (global outlier), or a group of data points in relation to all other data (collective outliers), or context-dependent outliers (e. g., failure rates of 5 % being acceptable for consumers, but not for enterprise customers) [HKP11, pp. 545ff.]. As with machine learning, methods to detect outliers can be supervised with expert input, run completely unsupervised based on math-

ematical definitions alone or be mixed, semi-supervised methods [HKP11, pp. 549ff.]. Besides statistical, proximity-based measures (e. g., Euclidean distance in two-dimensional space) to find outliers (cf. [HKP11, pp. 560ff.]), also methods based on classification and cluster analysis can be employed (cf. [HKP11, pp. 571ff.] and [HKP11, pp. 567ff.]). Further challenges arise for high-dimensional data (i. e., with many attributes), which needs specialized approaches [HKP11, pp. 576ff.].

Pattern, Association, and Correlation Mining. This category has a multitude of methods and applications for data analysis. While detecting frequent patterns, associations, and other correlations can be a goal in itself, the result of these descriptive activities are often crucial input for predictive models, as they capture knowledge about existing data (e. g., detected patterns can be the foundation for predictive classifiers). Typical results of **association pattern mining** (or frequent itemset mining) can often be encountered in online retail stores like Amazon. When customers surf to a product page of a gaming console, they are also presented with a list of games and other products that buyers of this gaming console have bought together with it ("Customers who bought this, also bought..." as known from Amazon[62]). This relates to the underlying issue of the detection of *frequent patterns*. These express which items are frequently part of the same transaction and thus bought together regularly [HKP11, pp. 243f.] [Agg15, pp. 15f.]. This widely used application is also called *collaborative filtering*, used in numerous Big Data recommender systems (cf. [CCS12, pp. 1169f.]). Here, based on the behaviors of other users or his own buying behavior, a new user is classified and receives new recommendations [HKP11, p. 319]. Related to this is *association rule mining*. The basic goal of it is to create rules in the form of $X => Y$. Here, a set X entails the occurrence of set Y, which could be shopping carts in a store. Usually, these rules are detected with a certain confidence and support (i. e., number of cases the rule was measured with) [Agg15, p. 16]. Notably, frequent itemset mining is an efficient first step to mine association rules (cf. [AIS93]). Three popular method categories for frequent itemset mining are candidate generation (e. g., Apriori algorithm), pattern-growth

[62] http://www.amazon.com.

(e. g., FP-Growth), and vertical data formats (e. g., Eclat). They can be used for both *pattern-based classification* and *pattern-based clustering* [HKP11, p. 280]. Finding frequent patterns is not limited to textual data, e. g. for mining in time-series (cf. [HKP11, p. 319]), but also in spatio-temporal, image data (e. g., object detection) or other multimedia data. Han et al. present a detailed roadmap to pattern mining research outlining various perspectives on the topic [HKP11, p. 280]. In general, these techniques find local patterns in data, which distinguishes them from a rather global statistical model [LL15, p. 617].

Dimension Reduction. Keeping the number of variables used for predicting or analyzing ("predictors") a value as low as possible is crucial for Big Data analysis, because interpretability is better with fewer predictors [LL15, p. 92]. Especially when a lot of data with high variety from various sources is available in high quantities, too many attributes become increasingly problematic. As Larose and Larose elaborate, summarizing Bellmann [Bel61], an increase in attributes necessitates an exponential increase in sample size [LL15, p. 92]. Even with Big Data sizes it is thus advisable to reduce the number of dimensions. Moreover, when all attributes from all sources are kept, it is likely that a new data source cannot deliver on all the attributes, hindering analysis [LL15, p. 92]. The underlying root problem to solve is that predictor attributes are (relatively) irrelevant for the outcome or interrelated (multicollinearity), meaning that they are not independent. In the latter case, several attributes express the same factor or component [LL15, p. 93]. For example, some attributes could inherently describe the same aspect of data, e. g. *repair price*, *oil change price*, and *maintenance cost* could describe the same overall factor *car running costs*. While dimension reduction techniques are usually not an end goal of data analysis, they are an important method that enables meaningful and interpretable analysis results. Principal Component Analysis (PCA) [LL15, pp. 93ff.] and factor analysis [LL15, pp. 111ff.] are two techniques to reduce dimensions. For instance, a PCA can identify those factors, which account for most variance in the data, i. e., factors with low explanatory power can be left out afterwards [LL15, pp. 93ff.]. To determine which components to reduce another set of techniques can be applied, e. g., the so-called Eigenvalue Criterion (cf. [LL15, pp. 102ff.]). KNIME evaluated

seven techniques for Big Data dimension reduction, achieving reduction ratios between 62 % and up to 99 % for a data set with more than 15,000 dimensions (i. e., attribute columns) [229]. Further, ur Rehmann et al. conducted a survey on dimension reduction and other compression techniques for Big Data [uRLA+16].

A fairly useful approach are so-called *ensemble methods*. They are neither analytics techniques nor algorithms that solve a problem directly. Ensemble methods allows to combine multiple models into one supermodel [Fra12, pp. 154f.]. Combining these models can be done using simple averaging or with more complex means, e. g. a decision tree where different regression models can be selected. For example, one linear regression model could fit customers between age 50 and 65, who have purchased more than three items in total. While younger customers between 20 and 35 are assigned a logistic regression model and all the other customers are covered by a neural network. A decision tree could guide this selection process.

A more thorough discussion of more specific data analytics methods and techniques form the fields of data mining, as well as machine learning and predictive analytics can be found in various literature sources available, such as [HKP11], [WFHP16], [LL15], or [Agg15].

2.3.4 New Analytics Domains

[CCS12] identifies four new analytics application domains ("technical areas" [CCS12, p. 1172], namely text, Web, network, and mobile analytics [CCS12, p. 1174]) which can be efficiently analyzed in the Big Data age besides "regular" data analytics. These fields are often interdisciplinary taking basic data analytics methods and applying both generic solution approaches as well as ones from their respective domains on the foundation of Big Data and novel technologies [CCS12, pp. 1174f.]. Each of these have unique technical opportunities and challenges [CCS12, pp. 1175ff.]. Another area, identified by Hu et al. [HWCL14, pp. 672ff.], is multimedia analytics.

Text Analytics

As pointed out in Section 2.2.1, unstructured and semi-structured textual content, e. g., e-mails, documents, Web articles, or social media posts, contribute greatly to the overall available Big Data. Text analytics especially deals with, among others, information extraction, topic modeling, opining mining, sentiment analysis, and text visualization [CCS12, p. 1174]. These methods and techniques are rooted in the fields of information retrieval and computational linguistics (e. g., NLP). They help to analyze contents of the text more deeply. This is particular challenging as natural language needs to be structured and "understood" so that information can be extracted from it. Sentiment analysis (also known as opinion mining) is a popular method to analyze user sentiments or opinions towards entities and their attributes, e. g., products and services or whole companies and organizations [Liu12, p. 1]. It is an umbrella term subsuming several tasks like opining extraction [Liu12, p. 1]. More specifically, e. g., posts on social media or Amazon reviews can be analyzed if they express a positive or negative sentiment towards a company's product. For that, various problems need to be solved, such as detecting and representing text structures through different languages, filtering irrelevant words or determining if a word or sentence is meant positively or negatively connoted in a particular context. These tasks should be done on great scale in an automated fashion [Liu12, p. 2]. The results and their understanding can help to build predictive classifiers, which can evaluate a new text for positive or negative sentiment. Especially reviews for products and service greatly influence customers buying behaviors. It is usual to evaluate other opinions on a product as base for one's own purchasing decision [Liu12, p. 2]. Besides analyzing customer reviews, e. g., [OBRS10] correlates public opinion from election pools to sentiments expressed via tweets on Twitter. [WCK+12] analyzed sentiments on Twitter during the 2012 US presidential elections. In general, analyzing short tweets (140 to 280 characters) poses challenges for techniques tuned for longer, fully formulated texts (e. g., cf. [KWM11]). Feldmann [Fel13] provides an overview of suitable techniques and lists further applications. An in-depth look into sentiment analysis can be found in various other sources such as [Liu12].

Web Analytics

Traditional uses of Web analytics were pioneered for Internet search, e. g., by the Google search engine[63]. Web sites needed to be scraped and analyzed, also including techniques from aforementioned text analytics [CCS12, p. 1176]. As Web 2.0 fueled Big Data development (cf. Section 2.2.1), new analytics opportunities arose as well. Especially a multitude of heterogeneous and rich Web sources, e. g., YouTube, Twitter, Google Photos[64], or Facebook is offered, most of them can be accessed via publicly available APIs and REST-based Web interfaces. This also includes various cloud services such as Google Maps[65] or image recognition services such Google's Vision API[66]. These data and services form the foundation for modern Web analytics [CCS12, pp. 1176f.]

Text analytics methods can be used for Web analytics, too. For example, in social media analytics, sentiment analysis helps to analyze positive or negative attitudes user posts. Apart from that, social mining and search are research fields related to Web analytics [CCS12, p. 1177] (cf. [ZAL14]). Another aspect is reputation systems, which can be found in numerous sites today [FG10, p. 18]. For example, reviews on Amazons display an information if and to which extend a review was helpful and users can actively mark reviews as such. On Steam[67], a computer games sales and management platform, users can also mark game reviews on their store pages as helpful or even "funny" (cf. [393]). Analyzing these systems, e. g., to determine users with high and low reputation (cf. [MAMASF13]), can yield further benefits and help to improve, e. g., search, social network filters, or recommender systems (cf. [FG10, p. 20]). Another facet of this topic is social marketing, which leverages social network for dedicated marketing efforts among various social media, such as blogs, social networks, or microblogging services (cf. [Zar09]). For instance, DeVries et al. study how social media

[63] https://www.google.com.
[64] https://www.google.com/photos/.
[65] https://developers.google.com/maps/.
[66] https://cloud.google.com/vision/.
[67] https://www.steampowered.com.

posts including images, videos, text, and links influences the number of "likes" or the user engagement via comments [DGL12].

Network Analytics

Leveraging latest growths in and spread of social networks, networks analytics aims to study the "dynamic nature" of these networks [CCS12, p. 1177]. Mining links between social actors (e. g., friend relationships on Facebook) or detection of particular communities is identified as research focus by Chen [CCS12, p. 1177]. However, analysis of networks grows beyond social networks, although the latter offer a vast body of data. Networks can also be composed of users, products, or services (cf. [RWE15]). Their linkage can be (friend) relationships, exchanges (e. g., mails or telephone calls), collaborations, and other interaction patterns [CCS12, p. 1177]. For instance, Kumar et al. analyzed certain social network communities and concluded that several segregated groups exist such as single users or isolated, star-shaped communities [KNT06]. With predictive analytics in this field, e. g., using the common neighbors technique or Jaccard's coefficient [CCS12, p. 1177], future links can be predicted. Clustering and machine learning approaches (cf. Section 2.3.2) are applicable to detect communities inside social networks (cf. [YHF10]). Depicting such networks as graphs has opened up the ability to use many known techniques for graph analysis on these diverse networks. Chen et al. note that network analysis techniques would not be part of most existing BI platforms [CCS12, p. 1177].

Mobile Analytics

Mobile analytics exploits the widespread use of smartphones and tablets. Their use has only increased worldwide since 2007. Statistics indicate that by 2020 more than 3 billion people will use smartphones globally [167]. On the one hand, smartphones represent a new kind of data source. They have various sensors and radios, such as GPS, gyroscopes, microphones, cameras, and WiFi and mobile broadband antennas. More than 5 million mobile apps (cf. [146, 104]) on market leader platforms Apple iOS and Android [367]

can, theoretically, exploit this data. They all can generate data and enable various use cases and analytics, e. g., for various fitness applications (e. g., cf. [YM15]). Another analytics-related trend is called "Mobile BI" [CCS12, p. 1178] (or: "Mobile Analytics" [173]), where BI reports and advances analytics are accessible via mobile devices. Additional methods especially include mobile social networks and social advertising, which can be analytically exploited with network analytics.

Multimedia Analytics

Multimedia data particularly comprises audio, images, and video files. Big Data analytics enables several forms of new data analyses on such heterogeneous data. Hu et al. mentions several research fields that emerged related to multimedia data: summarization, annotation, indexing and retrieval, suggestion, and event detection [HWCL14, p. 196]. For instance, audio summarization attempts to extract key phrases for audio files, whereas video summarization attempts to capture key frames from video files. For instance, Rahul Kindi provides a video summarization service, *vidDistill*, for YouTube[68] videos [227]. It uses a text summarization algorithm to summarize a given video into a desired text summary and employs video subtitles to match text passages to the correct section of the video. Hu et al. note that such applications were slow in general [HWCL14, p. 196]. Multimedia annotation is a related field, where labels are sought, which describe image, video, or audio contents and, e. g., allow better summarization [HWCL14, p. 196]. The research field of multimedia indexing and retrieval attempts to store and organize multimedia data by describing in a structured manner. After a first video analysis, techniques here include well-known data analysis techniques such as feature extraction (cf. dimension reduction in Section 2.3.3), classification, and data mining in general [HWCL14, p. 196]. Notably, features in videos also include detection of objects and other regions or scenes of interest [SZMR99]. Apart from that, [KL15] lists further methods, which can be used for multimedia indexing. Finally, multimedia data can also be used

[68] https://www.youtube.com.

for recommendations purposes, leveraging well-known algorithms such as collaborative filtering (cf. Section 2.3.3). These approaches also rely on a content-based analysis of multimedia data enabled by, e. g., aforementioned indexing and retrieval techniques [HWCL14, p. 196].

2.4 Process Modeling

Processes are a suitable way to represent *methods* as research artifacts. Like methods, processes depict a set of steps when viewed at a high level of abstraction. In the context of business processes, a process is defined as "completely closed, timely and logical sequence of activities which are required to work on a process-oriented business object" [Ros03, p. 4]. For example, a bank transaction involves such sequence of activities. A customer issues a transaction request, the bank system checks the transaction and then executes it, which involves contacting another bank's systems, before giving feedback to the customer.

Rosemann identifies several purposes of process models. These include, e. g., the *documentation of an organization*, *process-based reorganizations*, *knowledge management*, *software selection*, *model-based customizing*, or *simulation* [Ros03, pp. 43ff.]. For instance, a *documentation* purpose aims at creating transparency of documented processes, which allows to communicate them for, e. g., employee training [Ros03, p. 43]. In *model-based customizing*, reference processes are used to for "parameter-based configuration" of enterprise systems [Ros03, p. 46]. Here, a reference model depicts software functionality, which is configurable [Ros03, p. 46].

Process models are suitable to support the construction and visualization of a method as solution artifact as this work's contribution. This is due to the natural ability of processes to depict a method and possible advantages of process models in representing processes, such as creating transparency and allow for communication of said models — an activity, which is also mandated by the DSRM process (cf. Section 1.2). This section briefly introduces the basics of *Petri nets* as a process modeling language, as it is employed to model the GOBIA method.

As with models in general, process models can be depicted textually or visually as diagrams or drawings (cf. Section 1.2). In essence, process models are abstractions of the real-world concept of processes. Several *modeling languages* for (business) process modeling exist. In spite of this, there is no agreed-upon standard modeling language [Ros03]. Notable languages include Event-driven Process Chains (EPCs) [Sch02], Business Process Modeling Notations (BPMNs) [ISO13], and Petri nets. As Petri nets are used to model the GOBIA method, they are briefly introduced in the following. A more comprehensive view on Petri nets can be found in the literature, e. g., in [Rei85].

Petri nets are a process modeling language, which are focused on *information flows* [Rei85, p. 1]. They were originally proposed in the dissertation of Carl Adam Petri [Pet62]. Their concepts are rooted in network theory [Rei85, p. 3]. Each Petri net consists of two basic building blocks, namely *places* and *transitions*. Places are connected with directed *arcs* to transitions and vice versa. Figure 2.7 depicts a basic *condition/event* Petri net. Here, places are interpreted as *conditions*, and transitions as *events* (cf. [Rei85, p. 3]). "Data", "Processed Data", "Analyzed Data", and "Stored Data" are conditions, whereas "Process", "Analyze", "Store", and "Retrieve" are events. Conditions define which events can be triggered. Tokens in a place (e. g., as in "Data" in Figure 2.7)) represent a condition that holds, whereas a *set* of conditions that hold is termed *case* [Rei85, p. 3]. So-called *preconditions* are conditions before an event, which need to hold for the event to take place. *Postconditions* are conditions that hold after an event has triggered. In Figure 2.7, the "Process" event can only fire if there is a token in the place "Data".

More generally, not just one token can be in a place. In this case, the term condition might not be suitable any longer. Then, the generalized form as *place/transition* net is employed. This allows to interpret tokens individually as data objects. For example, if two tokens were placed in "Data" in Figure 2.7, triggering a "Process" transition would consume one token and result in one token in the "Processed Data" place while one remains in "Data" (cf. [Rei85, p. 6]).

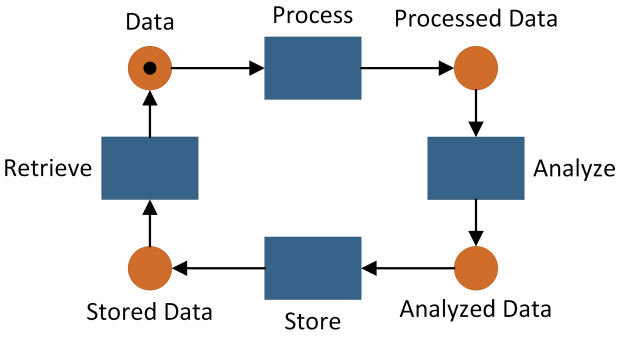

Figure 2.7: Cycle of data processing, analysis, storage, and retrieval as Petri net.

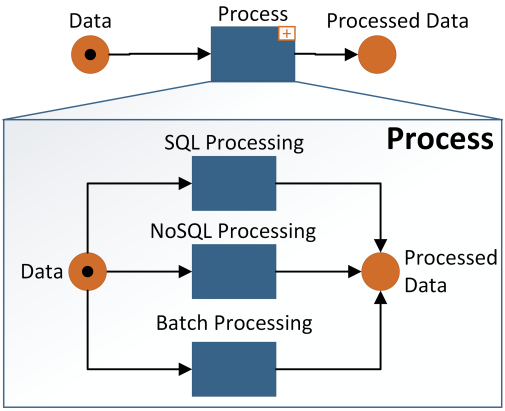

Figure 2.8: A Petri net refinement of the "Process" transition in Figure 2.7. To illustrate a refined transition, a visual indicator in form of a plus (+) sign is added.

Refinements, Logical Operations, and Notations in this Work. Petri nets allow to define arbitrary logical operations using an appropriate combination of places and transitions (e. g., cf. [Rei85, p. 9]). For instances, two transitions leading into one place constitute a logical disjunction (*OR* function). This also includes the possibility to define iterations. In addition to that, transitions in Petri net can also be refined [Rei85, p. 16]. In this case, a transition refers to a second Petri net model, which is responsible for the actual computation on the token that is emitted (cf. Figure 2.8).

In order to render the visuals of Petri net models clearer and more concise, certain modeling notations are applied when using Petri nets for modeling of processes in this work. These are derived from notations used in the Horus Business Modeler[69], a software tool for process modeling with Petri nets. In essence, *XOR inputs*, *XOR outputs*, and *optional inputs* are depicted using *fine-dashed* (XOR input and output) or using *coarse-dashed* lines (optional input). This is illustrated in Figure 2.9. Apart from that, start and end places (sources respectively sinks) are depicted using different colors and border styles than regular places.

2.5 Requirements Engineering

To attain a specific architecture for an organization, a process of designing and building it is involved. Often, reference architectures or other architecture templates[70] are used as aid in this process. The task to develop a customized architecture is done in business projects or within larger programs. Before the actual implementation, requirements of a to-be-implemented system are gathered. A project is implemented and executed according to these requirements (cf. Section 2.5.3).

Failure to properly capture requirements can severely hinder successful completion of software projects. A 1995 study by The Standish Group indicates that only 15 % of projects are completely successfully and within financial and temporal bounds. Approximately half of them managed to

[69] https://www.horus.biz/en/products/business-modeler/.

[70] These are detailed in Section 3.1.

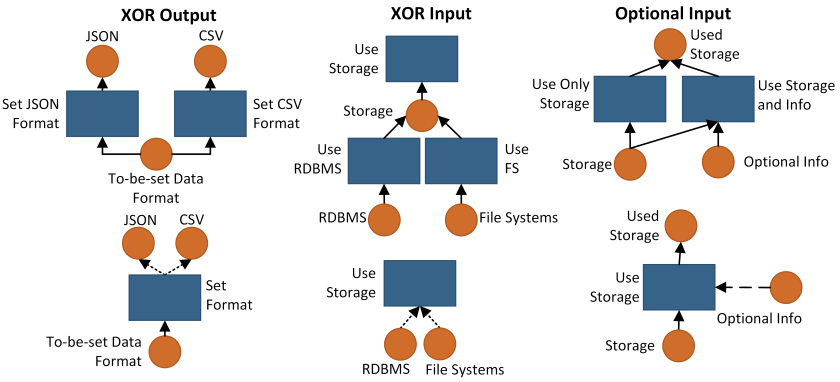

Figure 2.9: Petri net notations used in this work: XOR output, XOR input, and optional inputs. Top: traditional Petri net style, Bottom: Notation in this work.

be completed, but were delayed, have cost almost two times more than originally anticipated, and offer less features than planned. One third of all projects is actually canceled before completion[71]. The damages due to this are estimated to be more than 140 billion USD per year. [The95, p. 3]

Incomplete or changing requirements are among the two top factors that negatively impacted projects. One the other hand, clearly stated requirements as well as stakeholder involvement and support are major factors for project success [The95, pp. 8f.]. Notably, other factors such as technological complexity and maturity also contribute to successes and failures — as do project management and senior executive support [EK08]. This stresses the importance of requirements when the technological environment is complex, as it became by the emergence of Big Data technologies and products.

Other surveys indicate that 50 % of all defects found in software can be traced back to fundamental problems during requirements analysis and

[71] While more recent studies indicate an improvement by finding a larger portion of successful instead of failed projects, the amount of "challenged" projects has changed only slightly [400].

specification [Hen08, p. 24]. Henry notes that faults and defects detected during software testing reveal problems that can be traced back to respective requirements. In most traditional development models, requirement analysis is the first phase, while testing is one of the final phases [Hen08] (cf. Section 2.5.3. This underlines the importance of proper requirements engineering efforts.

Several aspects contribute to errors in requirements analysis, but most of them are human factors, which are not directly IT related. For instance, insufficient knowledge and underestimation of complexity can lead to faulty requirements (cf. [Hen08, p. 24]).

Defining Requirements and Requirements Engineering

The domain of *requirements engineering* is concerned with the "systematic and disciplined" specification and management of requirements [PR15, p. 4]. Their development encompasses elicitation, documentation, validation and negotiation, and management of requirements [PR15, p. 4]. According to Pohl and Rupp, these core activities lead to requirements engineering meeting its two goals:

1. Collection and standardized documentation of relevant requirements, aligned with stakeholders, and their systematic management. [PR15, p. 4]

2. Comprehension and documentation of stakeholder needs and wishes as basis for requirements specification and management to avoid results that are unable to meet these needs and wishes. [PR15, p. 4]

The word *requirement* itself is defined in the common standard ISO/IEC/IEEE 24765:2010. Essentially, it refers to a *condition or capability* or its *documented representation*. However, this condition or capability can be understood either as "needed by user to solve problem or achieve an objective" or something "that must be or possessed by a system, system component, product, or service" (i. e., essentially, an IT artifact) to fulfill an informal or formal constraint

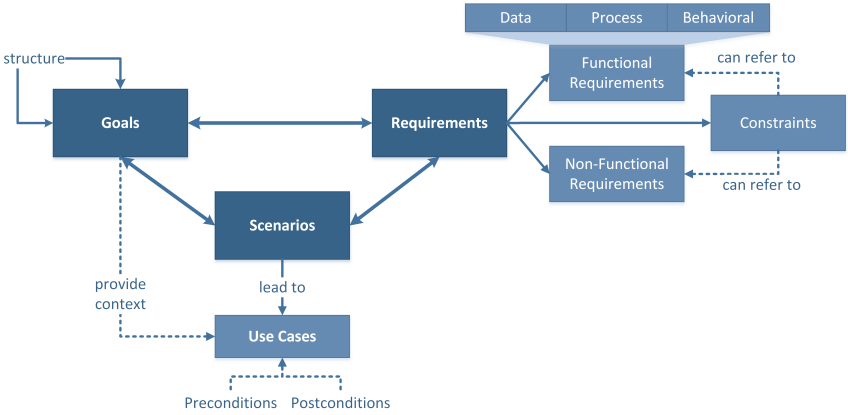

Figure 2.10: Goals, requirements, and scenarios as three main artifacts of requirements engineering, their relationships, especially to use cases, including a decomposition of requirements. Based on [Poh10, p. 43; pp. 17ff.; pp. 163ff.].

(i. e., an agreement, standard, specification, or other documented format) [ISO10, p. 301].

This acknowledges that requirements are not only needs and necessities regarding a to-be-implemented artifact. While capabilities express what functionality a system should possess, conditions impose quality criteria or other constraints and principles that steer specific behavior. This definition also refers to the users' perspective, who are key stakeholders of a designed system. Generally, a stakeholder can be any person or organization with influence on requirements. They are crucial to defining them as their needs and desires have to be met by the resulting artifacts [PR15, p. 3].

Pohl identifies three artifacts, which are the subjects of requirements engineering. Besides *requirements* there are *goals*, which must be fulfilled, and *scenarios*, which encompass interactions in the designed system. These are illustrated in Figure 2.10. Out of these, **goals** and **requirements** are detailed in the following sub-sections (cf. Section 2.5.1 and Section 2.5.2).

2.5.1 Goals

In the context of requirements engineering, Pohl defines a goal as prescriptive statement that captures stakeholders' "intention with regard to the objectives, properties, or use of the system" [Poh10, p. 53].

As Pohl elaborates, "goals drive and guide the elicitation of requirements" and provide the necessary background ("rationale") for requirements formulation [Poh10, p. 106]. In other words, goals can help to justify the inclusion and also exclusion of requirements into or from a requirements catalog. Specifically, for each goal one or more requirements can be defined that need to be satisfied until the goal is fulfilled. Pohl mentions use cases and scenarios explicitly as tool that aids in the formulation and understanding of each goal [Poh10, p. 149]. These use cases, a typical business tool, describe precisely how the system is supposed to work in conjunction with its task and end users respectively stakeholders (cf. [Poh10, p. 106]). Pohl notes that goals should be independent of the to-be-designed solution, i. e., "solution-neutral" [Poh10, p. 18].

However, the term is not exclusively used in requirements engineering, but in organizations in general. For instance, there can be higher-level goals related to corporate or departmental strategies [Poh10, p. 107] that do not directly concern a specific system. Ultimately, higher-level goals need to be refined into various more concrete sub-goals in order to be operationalized in requirements engineering [Poh10, p. 107]. This refinement task also falls into the realm of requirements engineering. These sub-goals can relate in different ways to their parent goal, namely with so-called "OR" and "AND" refinements. In the first case ("OR"), even one fulfilled sub-goal (of several others) leads to an overall satisfaction of the respective parent goal. This is equivalent to a logical OR composition of these goals. In the second case ("AND"), all sub-goals must be fulfilled for the overall goal to be satisfied as well. This equals a logical AND composition of the sub-goals. Pohl also documents relationships between different goals ("dependencies"). For instance, goals can support one another, needing other goals to be fulfilled as well, or be in conflict with those [Poh10, p. 108ff.].

Documenting Goals. There are various approaches to model goals in a more structured manner and to explicate its interrelation and its relation to specific requirements (e. g., cf. [LDDD91, DvLF93, HAC+16]). For instance, the modeling language MEMO GoalML distinguishes between "symbolic goal" (high-level) and "engagement goal" (low-level) [BF16]. In general, such approaches can be termed *goal-oriented requirements engineering* as these are focused on goal models [HAC+16, p. 107].

In addition to such visual, model-based approaches of documenting goals, goals can be described in a structured and text-based manner using natural language (e. g., cf. [Poh10, p. 115]). A sample goal documentation template proposed by Pohl, which denotes several phases of gathering information on goals, is listed in Table 2.8. Pohl argues in favor of a combined approach, where text-based documents are used in conjunction with visual models [Poh10, pp. 378ff.].

2.5.2 Requirements

Pohl synthesizes three types of requirements from various sources [Poh10, p. 17]. The two basic types are:

Functional requirements. These are the specification of "functionality" that a system provides to any user [Som10, p. 29] (cf. [Poh10, p. 17]). They describe what the system does in detail, thereby outlining inputs, output connection and the behavior of the functionality [Som10, pp. 29ff.] (cf. [Poh10, p. 17]). Based on this definition, three perspectives emerge according to Pohl: *process*[72], *data*, *behavioral*. A requirement may not necessarily fall into distinctively one of these perspectives, but may overlap between them. This point of view also influences the form of model-based documentation methods that are available for the requirements, as different modeling languages are appropriate for, e. g., data and process (behavioral) modeling (cf. [Poh10, p. 223]).

[72] Pohl refers to this as "functional", but essentially describes a process perspective (cf. [Poh10, p. 213]).

Table 2.8: Goal documentation template with attributes and descriptions. The recommended order of eliciting goal attributes is denoted in the "Step" attribute. The last three attributes are management information, which can be changed at any point. Attributes in *italics* are used for a streamlined documentation in this work (cf. Section 2.5.4). Source: based on [Poh10, p. 115].

Goal Attribute	Description	Step
Identifier	Unique value to identify goal, e. g. G_1	1
Name	Unique goal name, e. g., *The system shall provide customer response metrics*	1
Description	Comprehensible and understandable description of goal [Poh10, pp. 116f.]	1
Source	Stakeholder, document, or system, from where this goal was sourced	1
Responsible stakeholder	Stakeholder responsible for fulfillment of this goal	1
Priority	E. g., High, Medium, or a numeric scale	1
Super-goal	Reference to a super-goal identifier, if any	2
Sub-goals	Reference to any sub-goal identifiers, if any	2
Other goal dependencies	E. g., Conflicts between and indirect support of other goals	3
Using stakeholders	Stakeholders, who benefit when goal is reached	3
Associated scenarios	Identifiers of scenarios related to this goal	3
Supplementary information	Any relevant information not captured in other fields	3
Version	E. g., Version 1.0	-
Change history	Changes this goal documentation has undergone over time	-
Authors	Authors responsible for this goal documentation	-

Non-functional requirements (NFRs). According to Pohl, the term is misused for requirements which are improperly specified. He suggests the term *quality criterion* instead to underline that non-functional requirements (NFRs) are quality properties that need to be adhered to by to-be-designed artifact. While following this notion, this work will use the commonly used term NFR (cf. [Poh10, p. 19]). To specify quality, certain quality properties can be employed. For instance, Wiegers mentions several ones [WB13, pp. 197f.] such as availability, efficiency, flexibility, reliability, (re-)usability, maintainability, or portability. Such quality criteria can also aid to set acceptance criteria for requirements by defining specific quantitative or qualitative measures for these (e. g., a percentage for system availability).

Besides these two, a special, third type of requirements are **constraints**. These restrict the solution space, in which functional and non-functional requirements are realized [Poh10, p. 22] [PR15, p. 8]. Thus, they can specifically refer to both of the aforementioned requirement types, but also the development process of these as well. Constraints can be, e. g., hardware limitations (such as specific processors to be used), organizational rules (such as a four-eyes principle for server access), or regulatory standards (e. g., ISO 9000) [Poh10, p. 22]. While constraints are imposed by the organization, there are not limited to organizational constraints. Technological, economic, or regulatory constraints (cf. Section 2.6.2, [Poh10, p. 22]), such as environmental or reporting laws, may lead to constraint formulation. Notably, constraints may lead to a reformulation of other requirements, especially if these prohibit their realization [Poh10, p. 23]

These three types, two basic ones and constraints, are used for this work. Besides this, more fine-grained types of requirements may be appropriate, depending on the specific project (e. g., “informational requirements” as specific to Data Warehouse requirements collection in [SBL02]).

With respect to architectural designs, Pohl states that such activity depends on proper functional and quality requirements as well as design constraints [Poh10, p. 314].

Documenting Requirements

A straightforward way is to document requirements in a text-based manner, i. e., using natural languages such as English or German. In this case, requirements are described by words and sentences to detail their contents.

Following a strict requirements engineering top-down approach, various documents are created that describe the requirement in detail.

Compared to goals and use cases, requirements specifically describe "in far more detail" [Poh10, p. 217] target system properties or capabilities or conditions under which these should operate. Ideally, they are free of conflicts and unambiguous so that they can directly serve as basis for further planning or implementation (cf. [Poh10, p. 216f.]).

[Poh10, p. 300f] proposes the following criteria that each documented requirement ("requirement artifact" [Poh10, p. 16]) should adhere to. According to him, each requirement should be complete, traceable (to its source), correct, unambiguous, comprehensible, consistent, verifiable, rated, up to date, and atomic.

Besides identifiability, so-called "identification aspects" [Poh10, p. 340]), a mere detailed description and type of requirement (i. e., "content aspects" [Poh10, p. 343]) of a requirement, a structured Requirements Engineering (RE) approach can have several other requirement attributes stored with a requirement. For instance, Pohl mentions "context aspects", which include the source a requirement and potential stakeholders or management-related aspects such as a documentation of the development of a requirement or attributes that describe how the requirement is validated, i. e., checked for fulfillment (cf. [Poh10, pp. 340 - 348]).

Visual models can further complement textual descriptions, as with the documentation of goals. For instance, depending on the perspective to functional requirements several modeling languages are available to document these. Process models such as Petri Nets can be used to capture process requirements. Entity-relationship diagrams can be used to model data. Lastly, Unified Modeling Language (UML) state diagrams are able to depict systems behavior by modeling system states and possible transitions between them.

2.5.3 System Development Approaches

Requirements engineering is usually only one part of the development and implementation process of a software or a system. It is embedded in various ways in development approaches, which differ both in the positioning of requirement engineering in their development process, but also in their intensity of its usage.

One often used traditional systems engineering approach, which utilizes requirement engineering, is the so-called **V-model** [FM91] [WLRW15, p. 344]. Major contributions to this model are made by Frosberg [FM91]. However, newer variants evolved in Germany, which denote the V-model as "Vorgehensmodell" (engl. *procedure model*) (cf. [WLRW15, p. 344]). It is visualized in form of the letter **V** — hence the name — and starts with the elicitations of requirements, which are transformed and refine stepwise into a finished software specification, which is implemented by abiding precisely to these specifications. This first part is termed *decomposition and definition*. After implementation, the *integration and verification* phase begins. The implemented system is checked against the various specifications in reverse-order, i. e., the most technical and specific software specification comes first, before compliance against requirements and acceptance criteria is verified and the system is integrated [FM91, pp. 59ff.] [WLRW15, pp. 344f.]. In the V-model, the whole requirement documentation and software specification is completed, before the software is built according to these plans. Also, verification follows only a concluded implementation. The latest variants of the so-called "V-Modell XT" are published in [159]. Another notable plan-based approach is the **waterfall model**, which conducts software development in an iterative, but sequential manner. Here, requirements also come first, but after each phase (e. g., design, test, operation), it is possible to iterate back to the previous phase (cf. [Roy70] [McC96, pp. 136ff.]).

In contrast to that stand so-called **agile approaches**. These were fueled by the *Manifesto for Agile Software Development* by Beck et al. in 2001 [116]. Here, the authors call to de-emphasize rigorous usage of plan-based methods, strict apriori documentations and aim to focus on "responding to change" and delivering customer value by utilizing the strengths of individuals and

communications aspects [Mar03] [Col11, p. 8]. Agile approaches attempt to address perceived shortcomings of traditional approaches such as the V- or the waterfall model. These shortcomings include, e. g., discrepancies between captured requirements in the past and changed needs of the customer by the time of product delivery (e. g., cf. [Aug05]). Agile approaches assume that not all requirements and other details can be planned and captured ahead of time [JS12, p. 218]. Thus, they focus on "iterative", "evolutionary", and "incremental" executions, whereas iterations are kept deliberately short to counter the aforementioned issue. Additionally, agile approaches focus on *delivering value* [Col11, p. 8]. Processes supporting the development, just as any form of planning, documentation, or requirements engineering should be "barely sufficient" [Col11, pp. 7ff.]. Naturally, agile approaches also apply to the development of BI and other analytics systems [Col11] [KR13, pp. 34f.]. However, agile approaches are also subject to criticism themselves as, e. g., they could lead to the renunciation of any form of planning, documentation, or standards during systems development (e. g., cf. [JS12, pp. 222ff.]).

2.5.4 FROG AIR Case Example

Based on the aforementioned notions of requirements and goals, the specific goals and requirements for the FROG AIR sample case can be derived. The overall scenario was already outlined in Section 1.3.

More specific goals are derived from the overall goal for the BI system stated in the case scenario background. The initial version should serve as a prototypical implementation that focuses on the social channels from Twitter and Facebook. As the data for the social networks is available publicly on the respective platforms, that data can be utilized directly. The Customer Service Monitor should allow for a view across several channels — in alignment to the new omni-channel CRM strategy. Further requirements towards this part are elaborated upon, discussing combination of various Twitter and Facebook channels. Notably, the current working prototype does not implement all requirements at the point of writing of this work.

Goals and requirements are documented using a customized selection of the suggested documentation attributes in Section 2.5.1 and Section 2.5.2

above. Here, the focus is set on the *identifier* of the goals and requirements, their *names*, *descriptions* and *relations* to other goals and requirements. This focused selection is chosen to abstract from specific development approaches. In particular, this aims to allow using both agile and traditional approaches to define appropriate input requirements for usage in the contributed method. Further, unused metadata such as regarding stakeholders can be deemphasized, because it is assumed that the goals and requirements are aligned with those.

Goals

G_1: The system shall collect messages from specified channels.

Description: Necessarily, the channel that is basis for the analysis must be collected by the system in order to be analyzed. This does not dictate that the data must be persisted permanently, but the data needs to be in the CSM at least for the time of the analysis.

G_2: The system shall provide customer sentiments.
Dependency: G_1
Description: Sentiment analysis (cf. Section 2.3) should allow to capture whether the customer communication (initial answers and responses) have negative or positive emotions attached to it. This should be used as a quality indicator for the work in the channel.

G_3: The system shall provide customer response metrics.

Dependency: G_1
Description: Descriptive metrics on the response time of customer service agents to customers should be measured and provided. When the agents respond to a question on Facebook, the time discrepancy between their answer and the original customer request should serve as a key metric. Also other simpler statistics such as counting the number of interactions as indicator for channel activity should be provided. Lastly, as predictive metric, these numbers should be analytically correlated to the occurrence of public events. An indicator for the latter could be trending topics on Twitter.

G_4: The system shall allow for a comparison of the variously grouped sentiments.

Dependency: G_2
Description: Another goal is to be able to compare the previously provided customer sentiments per channel to one another. This should reveal additional insights into differences in customer sentiment and enable further investigations, if necessary.

G_5: The system shall rely on available technology as far as possible (in line with the company goal for software standardization).

Description: In line with the laid out strategic goal to reduce individual applications, the CSM should rely on software solutions from the software market and avoid individually programmed software where possible.

Naturally, some of these higher-level goals need to be decomposed into more specific sub-goals that relate more directly to the target system. For instance, G_2 needs to be broken down further to underline that the sentiments should be grouped by user ($G_{2.1}$) or by source channel ($G_{2.2}$), whereas both sub-goals must be satisfied so that G_2 is satisfied, too. Also, customer response metrics (G_3) need to be further broken down into a more specific metric, such as the average response time per channel or as the number total interactions per channel ($G_{3.1}$ to $G_{3.5}$). Only those goals at the bottom-most level, that relate more closely to the target system and accurately capture a stakeholder's intention can be used to more easily determine target BI requirements for the CSM (cf. [Poh10, p. 107]). Overall, the decomposition of these super-goals yields the following **sub-goals**:

G_2: The system shall provide customer sentiments.

- $G_{2.1}$: The system shall provide customer sentiments grouped by user.
- $G_{2.2}$: The system shall provide customer sentiments grouped by channel.

G_3: The system shall provide customer response metrics.

- $G_{3.1}$: Provision of average response times for each channel

$G_{3.2}$: Provision of number of interactions per channel

$G_{3.3}$: Provision of number of interactions per user

$G_{3.4}$: Provision of number of interactions by daytime

$G_{3.5}$: Correlation of the aforementioned metrics to public events (e. g., trending topics on Twitter)

After defining goals, scenarios can be used to give further details about these goals and make them more specific to the to-be-designed (cf. Section 2.5.1). The intended usage **scenario** for the CSM is that a responsible product manager for FROG AIR can navigate to it and pull the requested metrics and sentiment statistics for each FROG AIR channel. In the case of FROG AIR, there is not just one single Facebook page, where users can write entries onto, but there are several. As FROG AIR operates internationally, their language specific Facebook pages are separated from one another. Also, airline subsidiaries of FROG AIR that operate under a different brand name may also have a separate Facebook page. Each of these pages and Twitter profiles should be able to be analyzed, besides a total composition over all Facebook and Twitter pages. The user configures various display settings, such as selection-specific channels or users that should be included in the shown statistics. Ideally, the CSM provides a visual dashboard that can show various visualizations of the data, where appropriate. Also, the reports should work on recent data, i. e., it should be from the previous day at the latest. Moreover, performance of the dashboard should be within acceptable boundaries, i.e, response times should be in seconds span and not minutes span. Additionally, the statistical data provided by the CSM should also be easily accessible in a remote location, e. g., if MIDAS employees show the data to FROG AIR on their premises or even if third-parties should be granted access at a later stage. Thus, a Web-based access pattern seems favorable.

This leads to the **definition of initial requirements** for the CSM . These are based upon the criteria for defining requirements laid out before. Definitions will only include a focused set of aspects, such as an identifier respectively name and a description of the requirement including its relations to the goals and other requirements. Both functional and non-functional

requirements are included. Based on these requirements and the previously defined initial goals as inputs, the GOBIA.DEV process is started later. New information during the GOBIA.DEV process may lead to updated goals and requirements, respectively.

Functional Requirements

R_1: The system shall use user-defined Facebook pages for access.

Description: The user should have a visual way to select the Facebook pages which should be included into the comparison. This should not be restricted to a pre-defined set belonging to FROG AIR, but also competitors from the same industry (for instance) should be able to be added. For this, the URI part of the respective site should be used to identify the page.
Requires: —
Relates to: G_1

R_2: The system shall use user-defined Twitter sites for access.

Description: The same as with R_2, but with respect to individual Twitter profiles.
Requires: —
Relates to: G_1

R_3: The system shall provide access to Facebook data in the back-end.

Description: The system uses the publicly accessible Facebook API to read all messages from the sites specified in R_1.
Requires: R_1
Relates to: G_1

R_4: The system shall provide access to Twitter data in the back-end.

Description: The system uses the publicly accessible Twitter API to read all tweets from the sites specified in R_2.
Requires: R_2
Relates to: G_1

R_5: The system shall be able to disambiguate service agent messages from customer messages.

Description: From the data pulled from Facebook and Twitter, the author must be clearly distinguishable. In order to disseminate service agents from customers, the author is compared to the channel owner (Facebook and Twitter, respectively).
Requires: R_3, R_4
Relates to: G_1

R_6: The system shall determine customer message sentiments.

Description: A customer sentiment analysis shall be conducted on available customer messages.
Requires: R_5
Relates to: G_2

R_7: The system shall group customer message sentiments by user.

Description: The sentiments from R_7 should be groupable by user.
Requires: R_6
Relates to: G_4

R_8: The system shall group customer message sentiments by channel.

Description: The sentiments from R_7 should be groupable by channel (Twitter, Facebook).
Requires: R_6
Relates to: G_4

R_9: The system shall determine average response times by channel and arbitrary cross-sections.

Description: For each channel and subsets of these (i. e., cross-sections), the average response times should be calculated. The response time is the time delay (in seconds) between incoming customer service request and the first response of a service agent.
Requires: R_5
Relates to: G_3

R_{10}: The system shall count the number of interactions per channel.

Description: The system shall be able to calculate how many interactions (i. e., number of "back and forth" between service agent and customer) there are for each channel.
Requires: R_5
Relates to: G_3

R_{11}: The system shall count the number of interactions per user.

Description: The system shall be able to calculate how many interactions (i. e., number of "back and forth" between service agent and customer) there are for each user.
Requires: R_5
Relates to: G_3

R_{12}: The system shall count the number of interactions by daytime.

Description: The system shall be able to calculate how many interactions (i. e., number of "back and forth" between service agent and customer) there are for each daytime (i. e., by hour and by day).
Requires: R_5
Relates to: G_3

R_{13}: The user shall be able to visualize all data by bar charts.

Description: —
Requires: R_{7-11}
Relates to: G_{2-5}

R_{14}: The system shall be a user-accessible web application.

Description: A web-interface should be the system's graphical user interface. All interactions with the BI system take place there.
Requires: —
Relates to: G_5

Non-Functional Requirements

R_{15}: The response time for each request to the web application should be below 30 seconds.

2.6 Multi-Perspective Architecture Analysis

This section introduces the TORE framework for analyzing BI architectures and their construction process. First, PESTEL as a tool for analyzing the external environment of an organization is presented in Section 2.6.1. After defining the context of strategic management for such strategic analysis and positioning tools, the dimensions in PESTEL are detailed. Next, the TORE framework as a customized approach is built based on the idea of tools such as PESTEL in Section 2.6.2 and focuses on both internal and external dimensions.

2.6.1 Methods for Strategic Analysis

Strategic Management and Analysis. IS are an integral part of many organizations today. BI and analytics systems, being IS themselves, are, thus, by extension also crucial. While IS, which have gained wider traction since the middle of the 20th century, initially only had a supporting role in automating tasks or simple processes, more strategic potential has increasingly been recognized in recent decades (e. g., cf. [PM85]).

Especially analytic systems play a major role in this, as they actively support decision making with new insights (cf. Section 2.1). Having gained importance as (strategic) assets, they are now subject to strategic management (cf. [WP02, pp. 339ff.]).

With a business strategy, domains of future business activity of an organization are determined. For most strategies, both internal (i. e., capabilities of the organization) and external (i. e., the competitive market and its forces) contingencies are considered [Por79].

Strategic management is not limited to formulating such business strategies, but it is also concerned with implementing it inside an organization (as part of strategic planning) so that the implemented strategy yields the desired performance [WP02, p. 344]. Besides these aspects as part of a core understanding, scholars have discussed and see several other facets and perspectives to strategic management as research field (cf. [NHC07, pp. 946ff.]).

With respect to strategic management, the tasks and decisions made there, are thus larger-scale with long-term planning range (cf. [WP02, pp. 339ff.]). In contrast to this, tactical decisions refer to a medium-term temporal horizon and operational decisions are used in short-term for day-to-day decision and keeping the business running at the lowest scale (cf. [WP02, p. 1; p. 4]).

PESTEL. The Political, Economic, Socio-cultural, Technological, Environment, and Legal (PESTEL) analysis is an often-used analysis method in strategic management (e. g., cf. [Phi12, p. 74] [FN86, GJ90]). It focuses on a macro-economic analysis of the external contingencies of an organization. The result is an understanding of these external market forces, which are an input factor for strategy formulation and implementation [WH12, p. 101] [Phi12, p. 74].

A PESTEL analysis, as implied by the name, offers viewpoints for six external influence dimensions [WH12, pp. 101ff.], which are briefly illustrated in the following. The fact that PESTEL is often used and that it offers several dimensions as analysis viewpoints makes it one appropriate candidate to develop a customized analysis framework for this work in the next section.

> **Political.** Analysis of the political climate refers to factors related to governments policies. This dimension is situated between the legal and socio-cultural dimensions. Important factors to consider here are soft factors such as government stability and general attitudes towards native or foreign companies. Tax regulations are another factor to consider, not only from an economical point of view, but also with respect to complexity and accounting. Moreover, governments can offer special incentives, e. g., as tax exemptions. For example, Singapore offers such for newly founded start-up companies [214]. [WH12, p. 101]
>
> **Economic.** For the macro-economic environment, several indicators for assessment are offered. Firstly, there trends related to the overall gross domestic product (GDP) in the current location, which influences future supply and demand. Depending on the economic area, interest rates set by banks and federal reserves as well generic money supply and inflation rates are typical factors to consider (e. g., for investments

paid by loaned moneys or prices for leased properties). Other factors relate to the employment situation (i. e., unemployment rates) as well as related wages. Further, energy cost and availability can vastly differ for a location. [WH12, p. 101]

Socio-cultural. These factors influence both local employees as well as potential customers in a country. Age distribution of a populace in general and locally (e. g., many young or many old citizens), medical care and life expectancy, education influence both employment strategies as well products to be marketed. Moreover, cultural factors such as lifestyle of consumers and employees, their career paths, and family values need to be accounted. [WH12, p. 101]

Technological. Technological developments on a macro-level influence strategic planning. These include new products and other technological developments in general (such as Big Data) and their possible impact on productivity. In addition, technological infrastructure can drive strategic choices as well (e. g., cloud computing can make less sense if Internet speeds are poor). Spending into research from both governments and industry can fuel both market intensity and attractivity. [WH12]

Environment. This dimension refers to ecological aspects such as environmental protection regulations and norms [NB13]. Other factors include weather and other outside conditions in general. For example, Google exploits Ireland's cool climate to reduce cooling costs of its server farms [255]. On the hand, occurrence of natural disasters also influences strategy [NB13].

Legal. A dedicated legal dimension covers influences from various regulations, laws, and government-enforced policies on strategic planning. This includes tax laws, data protection laws, hiring laws, contract laws, or antitrust regulations [WH12, pp. 101ff.] [NB13].

PESTEL is an extension based on a set of related tools, which share some of its dimensions [NB13]:

PEST = Political, Economic, Social, Technological [FN86][73]

PESTE[74] = PEST + Environmental[75] [WH12, pp. 101ff.] (cf. [DP97])

PESTEL = PESTE + Legal

2.6.2 TORE Analysis Framework

The TORE analysis framework is proposed to offer perspectives on BI architectures and their development. It is inspired by the PESTEL analysis, but its factors are re-purposed for the context of this work. The analysis perspectives for architectures here require a different focus on the macro-environmental factors of PESTEL. First, as technology is a major aspect in this work, it is considered as first factor. On top of that, organizational and economic factors inside the firm influence the use and construction of architectures. Finally, this all happens within a certain regulatory environment. In addition to that, the TORE framework offers an *inwards* perspective on an architecture and is not primarily concerned with (external) macro-economic factors, although these may drive some of the internal aspects such as regulatory factors.

As outlined in the work's problem motivation in Section 1.1, novel Big Data technologies are often communicated with a focus on technology only. While such marketing strategy is purposeful for its beneficiaries such as vendors selling products or services on top of open-source tools, business decision makers and stakeholders implementing a BI reference architecture need to consider multiple facets of any technology that is to be used for an analytics project in their company. Thus, having a framework that offers analysis perspectives on BI architectures and their development, which is embedded in a holistic organizational context, yield several benefits. For instance, the development of a BI system, where an architecture has to be

[73] As SEPT.

[74] Alternative notations for PESTE are STEEP and STEPE [WH12, pp. 101ff.] [353].

[75] This is also referred to as *Ecological* [353].

constructed, is not only focused on technology, but also on organizational management process (e. g., project management or requirements engineering, cf. Section 2.5). Moreover, economic factors of employed technologies must be considered and weighed against technological properties. For example, it has to be verified whether the benefit of introducing a new technology outweighs the associated costs, e. g., relating to employee training, or to investments in new hard- and software systems.

Therefore, such new tool needs to focus on internal factors, while also considering external ones. However, the analysis perspectives do not need to be completely reinvented. As the previous section have shown, there are tools available which can serve as template. As BI systems gain strategic importance, it seems fitting to view them with dimensions derived from a strategic management context, although a BI system has — naturally — a different scale than a business strategy for a whole organization.

Technological. The technological perspective focuses on the characteristics of IT. It focuses on properties of the underlying technology, e. g., fault-tolerance, latency, or throughput — and other quantitative and qualitative properties. It is an inwards perspective on the organization, but fueled by new developments on the external market. On the internal side, already used technologies and needed properties can be matched by available technologies (e. g., any new Big Data technology such as Hadoop) and their specific properties. With respect to analytics and BI technology, most of it is software such as a database or a DWH. Nevertheless, hardware or appliances must be taken in consideration for certain type of systems, e. g., whether it is a single server or a distributed hardware system.

Organizational. Besides technology (IT), humans and tasks are part of a socio-technical system that is an IS (cf. [ÖBF+11, p. 8]). Like the previous perspective, this is an internal one. Any implemented software or hardware system, be it a BI system or another, is developed for and used by an organization or parts of it. Organizational aspects thus cover processes, structures, and resources (e. g., human resources, i. e., employees). An organization also imposes implicit and explicit restrictions and opportunities,

which must be considered when analyzing a BI system, be it a current system or a to-be-designed one.

Regulatory. The regulatory environment imposes structures, under which an organization operates. It represents an external influence factor. The previously shown models further distinguish these into more specific factors (e. g., political and legal). The most prominent regulatory influence factors are indeed the legal frameworks (i. e., the laws), which apply to the organization under consideration. These can be official regulations by the law-making body of a government, but also constructs such as an industry-consortium, which require their members to adhere to certain rules.

Economic. In the end, an organization has economic goals, which should be met. These economic goals directly relate to the chosen business strategy, e. g., cost-leader with low-cost products and a cost-efficient organization or quality-leader with higher-priced product with larger profit-margins. Also, any BI and analytic system to be designed, must meet economic goals aligned to these corporate economic goals. During an economic evaluation, naturally, direct costs are considered. In addition to this, several other factors need to be quantified, where possible. Especially for qualitative aspects or quantitative aspects without an economic measurement, the evaluation process can become more complex.

The TORE framework is primarily used in Chapter 4 to offer analysis perspectives for the constructed research artifact that illustrate its role and position inside a company and help to put it to use efficiently.

3 Analytical Architectures in Research and Practice

Having introduced the necessary technical backgrounds on traditional and novel technologies as well as data analytics itself — architectures — in particular analytical architectures, need to be analyzed and discussed in detail. These focus on the analytics functionality, which is a key component for decision making in BI.

This chapter details the complexity of analytical architectures that warrant the need for a universally valid BI reference architecture. Firstly, several facets of architectures and reference architectures are explained and detailed (cf. Section 3.1). To facilitate an improved understanding and to provide a structure to analyze architectures, the process of value generation in the Big Data age, so-called Big Data value chains, are analyzed and a unified value chain suitable to represent traditional and novel value generation processes using architecture is synthesized in Section 3.2.

Based on these two introductory sections, several types of architecture are detailed as related work and based on a literature search. An overview of advancements to Data Warehouses and their reference architecture is illustrated in Section 3.3. Then, based on the findings of a literature search (cf. Appendix A), identified Big Data reference architectures are discussed and characterized in Section 3.4. To augment this, a selection of identified practical usage of novel architectures is analyzed in Section 3.5, which result in additional architectural patterns. All this is used to motivate the need for a universal BI reference architecture, its customization, and to outline key components and patterns, which are utilized to formulate the research contribution.

With help of these findings and supplementary literature, several criteria according to the TORE framework are identified in Section 3.6, which guide the development and technology selection processes of analytical architectures.

Finally, all requirements to the contribution for this work are synthesized in Section 3.7. This is the *solution artifact* from the design science research process. This is done by summarizing the findings from this chapter, which outline the need for a universal BI reference architecture and a process to guide the development of a customized architecture including the required technology mix. These solution artifact requirements are used to formulate the GOBIA method in the next chapter.

3.1 Architectures Background

Before further discussing various types of analytical architectures and reference architectures, the term "architecture" must be defined and clarified.

Historically, the word "architecture" itself and its origin unsurprisingly predate any digital information system. In the physical world, an architecture can be the "complex or carefully designed structure of something", e. g., a building, or be used as term to summarize common design or construction patterns in real-world structures, e. g., *Gothic architecture* or *Postmodern architecture*. The latter definition is used in the context of a certain culture, place, or epoch [Hor00, p. 52]. Figure 3.1 illustrates these understandings of architecture using the Beauvis Cathedral in France as example.

The notion of "structure" in [Hor00, p. 52] brings the historical term closer to an understanding of architecture in the context of Information Systems. Specifically, the Oxford Dictionary [318] defines architecture as "conceptual" structure and "logical organization" of any information system or technology.

In the ISO/IEC/IEEE standard 42010, architecture is defined as "fundamental concepts or properties of a system in its environment embodied in its elements, relationships, and in the principles of its design and evolution" [ISO11]. The notion of *structure* and *logical organization* recurs in this defini-

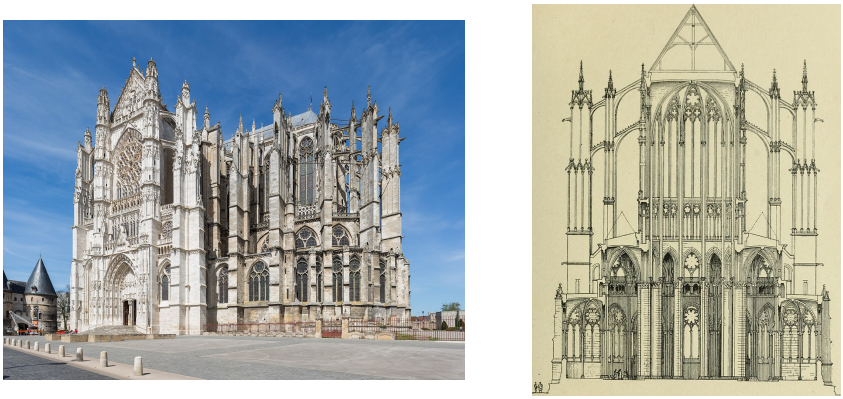

Figure 3.1: Beauvais Cathedral in Picardy, France. The photograph on the left depicts the cathedral as example for *Gothic Architecture* from the 13th century as construction and design pattern. The right image demonstrates the understanding of architecture as "structure of something" with a technical drawing (blueprint) of the side view of the same cathedral by Baudot. Sources: Left: David Illif [213], Right: [215, dB16].

tion. "Elements" and their "relationships" as well as "principles of [...] design and evolution" represent the structure of the system. The expression of important, i. e., "fundamental", "concepts" or "properties" inside this structure alludes to the abstract form that an architecture has, which is materialized by the aforementioned structure (cf. [204]). In the context of this work, a *system* refers to a *software system* in an information systems domain as hardware architectures are out of scope.

Strictly speaking, an architecture denotes a purely abstract representation of an analog or digital subject, e. g., a real building or a BI system. Formal visual or textual depictions of an architecture such as building blueprints are only one of several possibilities to model or to "express" this abstract architecture and to convey its meaning to stakeholders. These representations are referred to as *architecture description* (cf. [ISO11, 203]), although the term ar-

chitecture is often used synonymously to refer to an architecture description [203]. Figuratively, there is a difference between a BI system architecture in the mind of its creators and the visual models or textual descriptions that are created to represent it in various forms (cf. [203]). It is a typical feature of models in the Information Systems domain to be represented either textually or visually (cf. Section 1.2).

In essence, there can be more than one architecture description for an architecture (cf. [Gal15, p. 6]). Analogously several visual models representing the same architecture may exist. Each one might have a different view or perspective on the architecture. For instance, there could be multiple perspectives for different stakeholders such as business analysts, which focus on functional aspects, or programmers, who are interested in technical details [Gal15, p. 6]. The quality of a model is not just determined by its inherent properties, e. g., visual clarity [Ros03, p. 42], but also by the aforementioned fitness for use for its intended purpose, which also determines the viewpoint of the model. Models must be appropriate for their intended use to achieve a high quality. A model for a specific viewpoint can also be perceived as subset of a "total model" (as proposed for process models in [Ros03, p. 42]), which comprehensibly captures all elements and relationships.

In this work, the term *architecture* or *architecture model* is used to refer to a depiction of an architecture (i. e., one possible architecture description). In general, the distinction between architecture and architecture description is helpful for understanding the conceptual difference between the two. In this context, however, the term architecture is sufficient to capture the ideas that are to be conveyed without introducing ambiguities related to an "envisioned" notion of architecture. Potential discrepancies between forms of representation and the architecture they are supposed to represent, are not the primary focus of this work. It is assumed that an (architecture) model is one of several possible representations for an intended purpose, although the "fitness for use" of an individual model may vary in practice (cf. [Ros03, p. 42]).

Architecture Types

Consequently, a definition of *analytical architectures* is derivable from the generic architecture definition. Instead of referring to the concepts and properties of an arbitrary IT-related system, it refers to an information system whose *goal is to perform analytical tasks*. Analytical tasks can be comprised of traditional analytics (cf. Section 2.1) as well as advanced analytics (cf. Section 2.3). In that sense, an analytical architecture is a specific functional view of a generic architecture.

A *BI architecture* goes beyond analytic functionality as it refers to a holistic approach with a specific *purpose*. Nevertheless, analytics is often an important part of a BI architecture, because core BI functionality heavily uses analytics functionality. A BI architecture is the architecture of a BI system, which is an information system that contributes to BI in an organization, i. e., to the overall approach to offer business decision support (cf. Section 2.1).

A *Big Data architecture* is focused on representing Information Systems, which intends to specifically address Big Data challenges. It can focus on storage, processing, and analytics aspects, depending on its scope (cf. Section 3.2). Similarly, a *DWH architecture* is an architecture geared towards fulfilling goals of a Data Warehouse.

Naturally, there exists a subset of shared concepts among these architecture types. BI, Data Warehouse, and Big Data architecture may make heavy use of analytics functionality (cf. Section 2.3), which is the core viewpoint on an analytical architecture. As analytics is an important foundation of decision making, it used in this section to subsume various types of architecture for this purpose.

Reference Architectures

While a concrete architecture refers to a specific software system, a *reference architecture* is a system-agnostic *template* architecture. It combines several elements, from which a concrete architecture can be constructed. A reference architecture is an abstraction "from certain contextual specifics allowing its usage in differing contexts" [AGG12, p. 418]. This means that it resembles a

purposeful abstraction to allow utilization in different contexts other than the original one. It is not simply an arbitrary higher-level abstraction of one system-specific architecture. The template characteristic is a key determinant for a reference architecture and it is usually domain-specific [MvdL10, p. 30]. For instance, trivially, a BI reference architecture is a reference architecture in the domain of *BI*. A more specific domain might be a *Smart City BI reference architecture*, which is tailored towards BI applications in a smart city domain and its respective contingencies (e. g., IoT technologies).

According to Bass et al. [BCK03], software reference architectures are constructed using *reference models* and *architectural patterns* (see Figure 3.2). They also see it is a characteristic of reference architectures that they are used as a template for construction of concrete architectures. *Reference models* encompass a functional division of components, which cooperatively solve a specific problem in conjunction with flow of data between these components. This would be typical of mature fields, as experience is needed to build them [BCK03, p. 48]. For instance, DBMSs have widely known and agreed upon standard parts (e. g., a query processor; cf. Section 2.1.1). Together, the components of a DBMS solve the problem of managing a collection of formatted data in databases. As the examples illustrate, a reference model represents a decomposition of functionality. Brown et al. also note that a reference model should be perceived as consisting only of a "minimal set of concepts, axioms, and relationships" within the respective problem domain without distracting with too many details [BMH06, p. 4]. *Architectural patterns* contain not only elements and their relationships, but also impose *how* these are used, i. e., they describe interaction patterns between these components, which result in constraints on an architecture. Additionally, they serve as quality attributes for an architecture. It can be ascertained if and to which degree an architecture follows a certain pattern. Such patterns are often based on known solutions for a problem, which proved useful [BCK03, p. 48]. In software development, several design patterns have emerged to solve common programming problems (cf. [GHJV94]). For instance, an "observer pattern", related to publisher-subscriber patterns, specifies that the to-be-observed subject class notifies its observers in predefined situations and

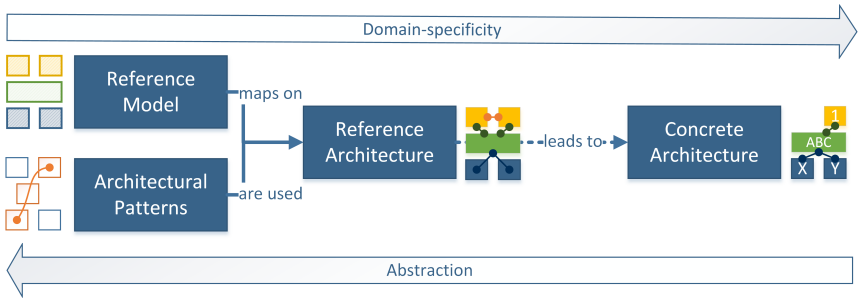

Figure 3.2: Conceptual relationships between reference architectures, reference models, architectural patterns, and concrete architectures. Two directed arrows indicate an increasing level of abstraction from concrete architectures to reference models, and an increasing domain-specificity the other way round. Source: based on [BCK03, AGG12, Gal15].

which kind of exchanges need to happen between them [GHJV94, pp. 293ff.]. In software architectures, a "client-server" pattern imposes certain roles and responsibilities for clients and servers and pictures the interactions between them, with the mode of interaction being of paramount importance. Whether a software follows a "client/server" or "decentralized" pattern has implications on the concrete software architecture and its implementation. A software reference architecture is seen as mapping of a reference model onto software elements, which implement both the specified functionality and corresponding data flows. [BCK03, p. 48]. Bass et al. interpret a software reference architecture in the sense of a system decomposition, where as a reference model is a division of functionality [BCK03, p. 48].

When analyzing work related to software reference architectures, Angelov et al. note that various works equate reference architectures with architectural patterns. Thus, they promote the aforementioned definition of Bass et al. [AGG12, pp. 418f.], which clearly distinguishes the two. It allows for architectural patterns to be viewed independently from their domains. In contrast to this, a reference architecture is geared towards a domain, which

is set out by the reference model (and the problem it intends to address). Architectural patterns can be used to construct reference architectures, which inhibit the desired qualities expressed in underlying architectural patterns [AGG12, p. 419].

Although serving as template to create custom architectures (so-called *facilitation*), Angelov et al. identify reference architectures for *standardization* purposes as a second variant of reference architectures. In architectures for standardization, the goal is to create a common understanding and naming of elements, concepts, and relationships as well as other inherent principles of a domain. The goal is to enable interoperability. However, this necessitates a certain maturity of the field as only commonly accepted elements should be incorporated for standardization [AGG12, p. 422].

Reference architectures can furthermore ensure a certain level of consistency of developed solutions based on them. They might further motivate an implementer to adhere to certain patterns (embedded in the reference architecture) or to related standards. Apart from that, reference architectures also serve other indirect purposes. For instance, they can provide a common language for project stakeholders to communicate or be used as point of comparison or validation of solutions against "proven" reference architectures [BCK03]. The United States Department of Defense (USDOD) sees a reference architecture as "authoritative source of information", which also functions as a guide for architecture implementers. They further explicate that a specific architecture is constructed from a reference architecture with the help of stakeholder requirements, which serve as design input in addition to a reference architecture [391]. Importantly, reference architectures can have different levels of abstraction as they cover a wide range between most abstract reference models and concrete architectures depicting a solution. Angelov et al. propose three levels of abstraction for categorizing reference architectures, namely abstract (e. g., an analytics functionality), semi-concrete (e. g., a class of technology options for each architectural element), and concrete (e. g., a specific technology choice out of several options). Notably, "concrete" should be interpreted in the realm of reference archi-

tectures and does not relate to the realm of concrete architectures [AGG12, p. 421].

Architecture frameworks are usually more domain-independent (i. e., details are abridged so that they are less domain-specific) and aid the description of reference architectures ("how to create and capture system designs" [MvdL10, p. 25]) by stating which information, structures, and form of presentation should be employed [MvdL10, p. 25]. Notable examples for such include TOGAF [The11] and the Zachman framework [Zac87]. *Architecture methods*, similar to the aforementioned, focus on guidance and a list of ordered steps from start to a finished design [Gal15, p. 6], which is often purposefully left unspecified in architecture frameworks [MvdL10, p. 25]. Using this interpretation, aforementioned reference architecture by the USDOD [391] also contains elements from an architecture method as it intends to offer guidance for architecture creation as well.

These defining characteristics of reference architectures by Bass et al. — with additions by Angelov et al. — can be readily applied to BI reference architectures (respectively DWH or Big Data reference architectures). Here, an architectural pattern depicts a fundamental structure of a solution to an analytical problem (e. g., analysis of semi-structured data). These can be distilled from, e. g., actual architectures from established solutions, documented best-practices, and technology characteristics (cf. [AGG12, p. 418f.]). Such reference architectures are also a mapping onto specific "software elements" that implement functionality from a reference model. For instance, the DWH reference architecture deals with classes of solutions that are implemented by software (e. g., "ETL" or "DWH" as abstract technology classes dedicated to a certain functionality). It is independent from a specific DWH, but still applicable to classes of problems a DWH intends to solve (e. g., as per the DWH definition, cf. Section 2.1.2).

Although certain architectural patterns for Big Data have already emerged (e. g., distributed Big Data processing with MapReduce), a more comprehensive view on architectural patterns in a Big Data context require further exploration due to the novelty of field. This is conducted through an overview of related work in Section 3.4 and Section 3.5. However, such patterns for

a traditional DWH are well known. For instance, the flow of data, and especially the value-adding enrichment through ETL and the creation of data marts respectively OLAP cubes, are typical interaction patterns in a DWH. Recent additions to and advancements of this are presented and analyzed in Section 3.3.

3.2 Big Data Value Chains

As with classical technologies, deriving insights and act upon them is also an important goal of analytical and Big Data architectures. An insight or any other use for an analytical output, is something of value to the respective stakeholders and the involved organizations. Consequently, a value generation process also exists in the realm of Big Data architectures, as it does in traditional Data Warehousing. However, while value generation in a DWH has been discussed for a prolonged time and the process is well-known (cf. Section 3.3), this is not yet universally the case for Big Data architectures. The model describing a value generation process is also commonly called *value chain*. More specific conceptualizations are termed *Data Value Chain* (cf. [Cur16, p. 31][163]) or, even more specifically, *Big Data Value Chain* (cf. [CML14, HWCL14]).

In general, a value chain describes high-level macro-tasks, which includes the sequence of actions necessary to create an item of value. For a Big Data or data value chain, the item of value is typically information, which is generated from raw source data and represents an insight for the involved parties. Such value chain especially describes how the source data is transformed into valuable information. Based on this new information, business decisions can be made (cf., e. g., [Cur16, p. 31]).

Several authors from both practice and academia deal with the topic of Big Data value chains — as illustrated in the upcoming section. However, there is no consensus on a common value chain. This may be due to the novelty of the Big Data field and the diverse set of both technologies and use cases, which the former enables. In spite of this, there are several commonalities between multiple proposals. After shortly describing a subset of these value

generation process proposals, a *Unified Big Data Value Chain* is proposed, which is based on the aforementioned. It aims to bring together the various facets of Big Data Analytics (e. g., batch analytics and stream processing) and also represent the existing, traditional DWH value chain (cf. Section 2.1).

Any BI technology or system, which executes the tasks described in a value chain, is designed to fulfill specific goals and requirements. These can be also be formulated as business questions to be answered by such a system. The answer to these questions is given using data. Naturally, value generation can be described more easily if both goals and data, which are needed to fulfill these goals, are fully known and available. However, this may not always be the case. If the goals are formulated, but not all data sources are known or available, additional measures have to be undertaken to discover and procure or generate all data needed. If several data sources are available, but no business goals are formulated that explicitly exploit them, required measures tasked with finding business goals need to be executed. Exploratory analysis is a typical tool to discover unknown insights or value inside data. Notably, a system that enables exploratory analysis is different from a system which builds on the business goals which are formulated because of the results of the exploratory analysis. Exploratory analyses serve to dynamically work with several different data sources and find such insights that lead to business goals for some of the data. Naturally, if both business goals and data are unknown, there is no business case to be solved. A value chain may incorporate explicit or implicit assumptions on the availability of business goals and data. A categorization of the presented value chains and the proposed unified value chains is illustrated in Figure 3.3.

3.2.1 Analysis of Existing Value Chains

In the following, a selection of five proposed Big Data value chains is discussed and compared. Their differences are pointed out and their fitness to represent "all" major types of Big Data use cases are assessed. Based on this, a unified view is formulated in Section 3.2.2, which aims to overcome the outlined views on Big Data value generation. The goal is to describe a Big Data value chain that is applicable in all (Big Data) use cases. This helps to

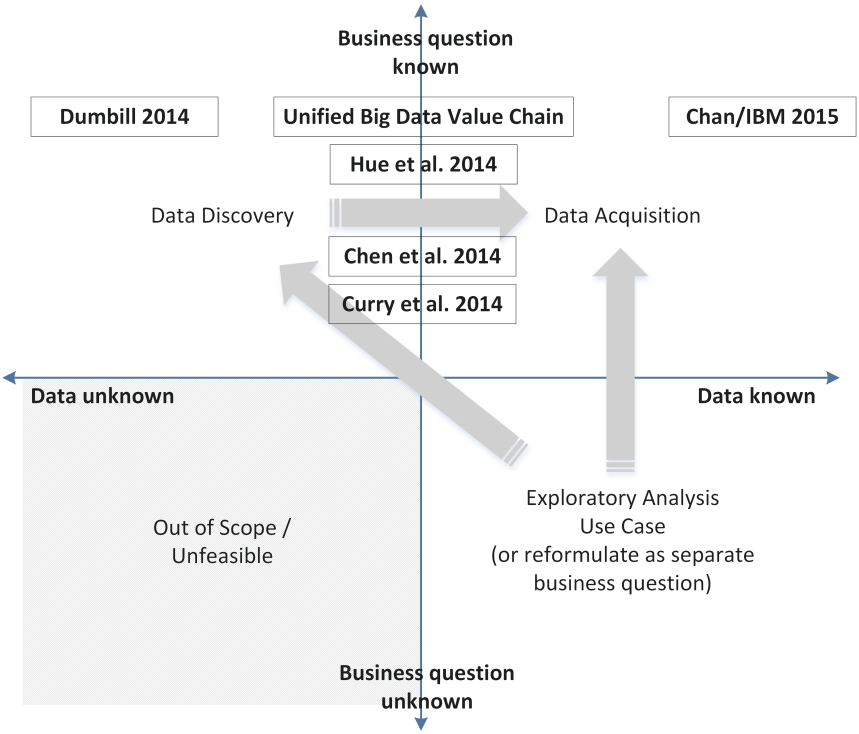

Figure 3.3: Categorization of Big Data Value Chains discussed in this section according to the extent underlying business questions and data are known ex-ante. The categorizations are not necessarily binary, but rather continuous. The matrix also details what kind of task needs to be done before data can be further used and analyzed.

Figure 3.4: Big Data Value Chain for Streaming Applications based on [131].

understand how typical architectures in this context work from a high-level view. This can aid to comprehend other processes and patterns with respect to traditional and especially novel Big Data aspects. That way, such a unified value chain can serve to analyze and structure both novel and traditional architectures from the perspective of their value generation.

Streaming Application Patterns by [131]

Although [131] is mainly a documentation for "IBM InfoSphere Streams", it still describes typical patterns for stream analytics applications as identified by IBM's Samantha Chan (cf. Figure 3.4). The depicted pattern constitutes a typical value chain. Its analytics part is a sequential pattern that consists of Data Ingestion ("Ingest"), Data Preparation ("Prepare"), and Data Analysis ("Detect & Predict"). This is followed by "Decide" and "Act" steps, which make use of the analysis results in a business scenario. As typical for stream processing with real-time or low-latency requirements, storage is seen as optional. It is not placed along this main analytics sequence, but only alongside it, where persistence is needed, e.g., for analysis later on. (cf. Section 2.2.6). Moreover, the authors see all layers except "Ingestion" as optional for streaming applications in general. However, this not sufficient for analytic applications, as an execution of analysis functions is needed, which are usually not situated in an ingestion layer. In this value chain, the "Ingestion" layer is only responsible for data acquisition, i.e., making streaming sources physically available for the stream analysis pipeline. For the data preparation phase, typical pre-processing, integration, and cleansing activities, as known from ETL, are deemed relevant for streaming, too

Figure 3.5: Data Value Chain based on [163].

("parse, transform, filter, clean, aggregate, or enrich [...]" [131]). However, this activity is seen as an in-memory activity, because the streams should usually not be saved on secondary storage to allow real-time analysis. Due to this, the specific tools are different for streaming cases (e.g., IBM advocates InfoSphere Streams for this task). The subsequent analytics phase is focused on real-time analytics, which fits the demands of streaming data, where often advanced and predictive methods are advocated, such as topic modeling and forecasting.

Data Value Chain by [163]

In a blog post for the *IBM Big Data Hub*, Edd Dumbill outlines a generic "Data Value Chain", which is, however, defined in light of the "excitement around new data technologies", i. e., the current age of Big Data with various emergent technologies. It views data as raw material, like "crude oil", which can be exploited and refined to create a more valuable insight (analogous to "refined oil"). Here "many steps of processing and combination" lead to value generation in raw data, described in a value chain with seven "stages" [163]. Dumbill places discovery of data sources as first step in his value chain ("Discover"). This is especially crucial considering that internal, and many more external, data sources become exploitable with new technologies (cf. Section 2.2). Further, found data sources need to be assessed in terms of "cost, coverage, and quality" [163]. Next, an ingestion step follows, where data is physically made available to the BI system. This step is similar to the corresponding step from the previous streaming value chain by [131]. The next two steps, "Process" and "Persist", represent typical ETL tasks. First, data needs to be brought into the proper target format of the system with means of pre-processing. Then, it is stored ("persisted") in a suitable data storage, where available technologies, as Dumbill acknowledges, are as diverse as

Big Data can be nowadays. In contrast to the streaming value chain, the author sees data storage as obligatory step ("every step in this chain is vital" [163]), with no explicit mention of persistence being optional. Interestingly, the task of data integration is seen as separate task ("Integrate"). Rather than being part of the "Process" step, the combination or correlation of various data sources is separate, which differs from traditional ETL processes, where these steps are combined. Unsurprisingly, data analysis ("Analyze") is the main part of this value chain, where the "so-called data janitorial work" would make up large portions of the data science field [163] [vdA14], with the goal of deriving new, actionable insights from data. Similarly to the streaming value chain by [131], propagation of the acquired knowledge back into the organization is explicated, although Dumbill does not explicate the impact of the data analysis on the organization at all ("The results of analytics [...] are exposed [...] in a way that makes them useful for value creation [...]" [131]).

Big Data Value Chain by [CML14]

[CML14] also outlines a "Big Data Value Chain". According to the authors, it consists of four sequential phases. "Data Generation", "Data Acquisition", "Data Storage", and "Data Analytics" (see Figure 3.6). In their model, Data Generation refers to the origin and properties of the data to be analyzed (cf. the "the Vs" in Section 2.2.1). They also point out and confirm that Big Data available today, e. g., IoT data or social media data from the Internet, "far surpasses the capacities of IT architectures and infrastructures of existing enterprises" [CML14, p. 179] (cf. Section 2.2). As for the Data Acquisition phase, they attribute the sub-tasks "data collection, data transmission, and data pre-processing" to this phase. The first task is "to acquire raw data" from the respective data generation origin [CML14, p. 181]. The task of Data Transmission is to transfer data to a "data storage infrastructure" [CML14, p. 182], from where it can be further processed and then analyzed. Notably, the authors see this central storage step as necessary for their value chain, while the first streaming focused value chain by [131] does not. The focus of Chen et al. in this step is to discuss the challenges of data transportation

Figure 3.6: Big Data Value Chain as both in [HWCL14] and [CML14].

between and inside various data networks, e. g., inside a data center and to other data centers. Their description of Data Preprocessing largely matches typical preprocessing tasks, notably integration and cleaning. However, Chen et al. imply a varying degree of necessary preprocessing ("*some* [...] methods have [...] requirements on data quality" [CML14, p. 183], "we shall pre-process data under *many* circumstances [...]" [CML14, p. 183]). They see this step as mandatory in general, even with Big Data, in order to reduce data size and increase analysis accuracy [CML14, p. 183].

Big Data Value Chain by [HWCL14]

[HWCL14] describe a sequential value chain with phases and map a Big Data technology timeline to these. Hu et al. see four phases in a "data value chain" [HWCL14, p. 657]: "Data Generation", "Data Acquisition", "Data Storage", and "Data Analysis" (see Figure 3.6). According to them, activities of a Big Data analytics system would "usually" fall into these four phases, aiming to capture the life-cycle of data from "birth to its destruction" [HWCL14, p. 656]. The overall model contains the exact same four key phases in the same order as the one by [CML14] discussed above. There are also commonalities in the description of the phases themselves. Data Generation is seen similarly to [CML14]. This phase is used to describe the inception of the data at hand and its properties depending on its origin, such as velocity (streaming or stationary data) or volume. In the Data Acquisition phase, Hu et al. see three sub-tasks: "Data Collection", "Data Transmission", and "Data Preprocessing". On this level, it matches Chen et al. as well. The goal of the collection task is to retrieve data "from real-world objects" [HWCL14, p. 661] and feed it into the target system. They see a clear correlation between the collection method and both properties of the data (from the Data Generation phase) and the objective of Data Analysis, which is conducted

further down the pipeline. They imply that the collection method must be aligned to both origin and target of the data in the to-be-constructed analysis pipeline. Also, they recognize that this leads to several data collection methods, whereas they see sensors, Web logs, and Web crawlers as "three common" and representative data collection methods deserving to focus on [HWCL14, pp. 661ff.]. Regarding Data Transmission, transfers inside a data center and between data centers, i. e., over the Internet, are to be considered [HWCL14, pp. 663f.]. Data Preprocessing encompasses typical preprocessing tasks such as integration, cleansing, and redundancy reduction (e. g., deduplication or compression) [HWCL14, pp. 664f.] - as in the chain process by Chen et al. (cf. [CML14, p. 183]). Similarly, Hu et al. recognize that there is no "one-size fits all" data preprocessing technique or method, but that one must choose the most appropriate ones for the data properties and the use case requirements at hand, e. g., regarding latency or the type of analysis [HWCL14, p. 665]. For them, Data Storage comes right after Data Acquisition and Preprocessing to store the "collected information" [HWCL14, p. 665]. Two aspects of storage are discussed for this. First, there is the physical aspect of storage, which concerns the underlying hardware infrastructure along the memory hierarchy (e. g., from classical hard drives to in-memory storage) and more sophisticated systems (e. g., a SAN). Secondly, the software level with data organization (e. g., traditional databases, NoSQL data stores, or distributed file systems), including query and other access interfaces (termed "data management"), are seen as important aspect there [HWCL14, pp. 665ff.]. Finally, Data Analysis can be conducted upon the preprocessed and stored data. In their value chain, Hu et al. identify several commonly used Data Analytics types (from descriptive to prescriptive analytics; cf. Section 2.3) and techniques (e. g., structured analytics such as OLAP in an RDBMS, text, or Web mining; cf. Section 2.3) [HWCL14, pp. 672 ff.].

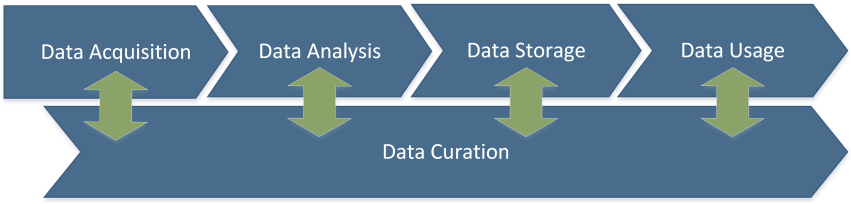

Figure 3.7: Big Data Value Chain based on [Cur16]. The original illustration features a purely sequential flow. Using the detailed textual descriptions of the value chain interactions, a more dynamic process is illustrated here.

Big Data Value Chain by [Cur16]

Another Big Data value chain is proposed by Edward Curry in [Cur16][1]. Curry structures his proposal into five "key high-level activities", which "comprise an information system" [Cur16, p. 31]: "Data Acquisition", "Data Analysis", "Data Curation", "Data Storage", "Data Usage". Notably, while the activities are originally illustrated in this sequential order (cf. Fig. 3.1 in [Cur16, p. 32]), the description by the author suggests a more complex semantics, which is illustrated in Figure 3.7. For Data Acquisition, the author sees "gathering, filtering, and cleansing" of source data as major subtask before it is stored (e. g., in a DWH or a "big data-capable storage" [Cur16, p. 40] like HDFS). Like the previous proposals, Curry also considers the various kinds of Big Data as input and the infrastructure challenges related to data properties. The infrastructure of the acquisition pipeline should also enable efficient querying of data, besides its capture. However, it does not necessarily mean that data has to be physically stored. In case of streaming, publisher-subscriber patterns can be employed to "query" the captured data [Cur16, p. 51]. Curry further underlines that choosing a proper acquisition tool, which fits the data properties and the respective use case requirements, is the main goal when deciding for a "data acquisition strategy" [Cur16,

[1] Curry recaps his own model here, which he originally proposed in 2014 in the context of an EU project "BIG".

p. 52]. In this value chain, regular Data Processing is seen as part of analysis, while preprocessing is seen as part of the acquisition phase. A more significant deviation of previously discussed value chains is the Data Curation phase. While it has some overlap with the cleaning tasks in the Data Acquisition Phase [Cur16, p. 51], it is deemed a supporting activity through the whole value chain ("active management of data over its life cycle" [Cur16, p. 32]). Therefore, it is not necessarily in a fixed place between Data Analysis and Data Storage, but can be regarded as an activity parallel to the other key activities. Indeed, the author outlines that curation activities could be seen from both data origin side (where reusability should be maximized and quality of the data with lowest possible effort) and from data consumption (i. e., analytics) side (data should fit the case requirements only) [Cur16, p. 91]. This notion is further mirrored by the typical distinction into exploratory/iterative and deterministic (i. e., traditional analytics according to a priori requirements) analyses in analytics use cases. The goal of this is to ensure that data, at all points in the value chain, fits the underlying requirements regarding its veracity, availability and accessibility, efficiency (i. e., avoiding redundancies), and quality [Cur16, p. 32]. Curry allocates responsibility for the underlying sub-tasks to "expert curators", i. e., humans.

3.2.2 A Unified Big Data Value Chain

To find a common view on a Big Data value chain that covers "typical" Big Data use cases, these five proposals are used to synthesize a *unified Big Data value chain*, which is illustrated in Figure 3.8. To formulate this unified value chain as common ground, the differences between the proposals need to be bridged. Thus, notable ones between the four introduced value chains are outlined and a common denominator is developed. After that, the details of the unified models are shortly described.

Each of the previous proposals made certain assumptions, both in the visualizations of their value chain and also in the accompanying textual description. One notable difference is that the model by [131] sees Data Storage as completely optional activity that is only invoked when needed. The other proposals both have an explicit Data Storage phase at a fixed

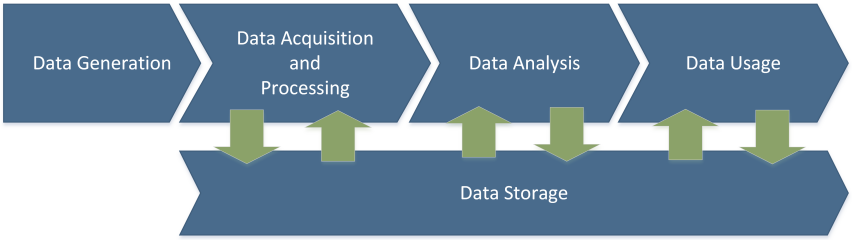

Figure 3.8: Unified Big Data Value Chain based on [131, CML14, Cur16, HWCL14].

position in their value chain, although the textual descriptions of some seem to imply that the storage function can be more flexible than that. To represent both streaming use cases with potentially low storage involvement and batch-based scenarios with storage involved at some point, the unified view represents Data Storage as optional supporting activity while indicating typical interactions with other phases (i. e., data can be stored after it comes into the system and before and/or after Data Analysis). It is through the necessities of a specific case that storage becomes mandatory.

Another notable difference between the proposals is the exact scope of and distinction between *Data (Pre-)Processing* and *Data Acquisition* (or "Ingestion") and their relation to the source data, often encapsulated in a *Data Generation* phase. The specific sub-tasks associated with these activities are, however, comparable. These are, first, describing the source data characteristics and transportation character of it (push vs. pull). Then, data is physically made available ("acquired") to the Big Data system by means of transportation. Following this, all proposals see some form of (pre-)processing to make the data fit for the analysis ("preparation" [131], "curation" [Cur16], or as unnamed part of a Data Acquisition activity [HWCL14, CML14]). After acquisition and pre-processing, an interaction with a storage system may occur. Alternatively, the Data Analysis activity follows. Similarly, this can be followed by storing analysis results or directly invoking the Data Usage activity, which is a generalization of various usage possibilities depicted in the presented models. This value chain is also flexible enough to cover the

value generation in traditional DWH, as it covers the basic activities as well, although in a more abstract fashion.

The separation into five high-level activities is done by isolating the respective **purposes** and associated tasks for the value chain. It transforms data by means of analyses and the other phases from raw data into information.

Notably, this categorization is employed to derive a structure for various novel and traditional tools in Section 4.2 and, afterwards, to formulate a selection process for these in Section 4.3.

Data Generation. The purpose of this activity is to clarify the technical data properties (velocity, variety, volume) and to further specify the data source characteristics. In particular, whether push or pull mechanisms are to be employed to access the raw data, how trustworthy ("veracity") it is, how high or low its potential business value is at the moment, and how high its potential value is. This information guides the value chain in which actions to take and which, e. g., technological, choices to make later on. For instance, a real-time data stream that uses push-mechanisms and needs real-time replies will lead to other methods for data acquisition (e. g., without heavy pre-processing) and suitable methods for data analysis and only non-obstructive forms of storage, if any at all before analysis, to achieve the goal of low latency. The term *Data Generation* refers to both known and yet-to-be-discovered data sources, as it means that somewhere data is generated at some point. While it presumes that at least some data is known ex-ante, data discovery is meant to be conducted in this phase on-demand.

Data Acquisition and Processing. The Data Acquisition phase is divided into two major subtasks. First, in the **Acquisition** part, raw data is made available to the BI system. Either the data source is actively accessed ("pull" mechanism) or data sources itself actively "pushes" data to a target sink (e. g., via a publisher-subscriber pattern [Psa17, pp. 20f.]). A sink could simply be a service or tool that receives data via a public URI. Pull mechanisms are often used for batch-based analytics, which can be traditional DWH scenarios or MapReduce applications. A typical form of active pull-based access would be an ETL tool that retrieves data from an OLTP RDBMS via SQL queries

in order to integrate it into a DWH. Push mechanisms, on the other hand, are typical for streaming applications, where there is no finite set of data which can be "pulled". The second part is **Processing**, which follows the Acquisition part. Generally, "processing" refers to any transformations of data into the form that is needed by a subsequent activity. Often, the term *processing* is meant to be understood as *pre-processing* as means to prepare the data for later analysis (and, implicitly, storage, if needed). However, the term *processing* is chosen as name for the phase, because it is more generic and could, e. g., allow for different kinds of invocations such as *post-processing* after analysis. In Big Data scenarios, the degree of pre-processing is highly variable and dependent on the use case (in contrast to traditional scenarios, cf. Section 2.1). Some streaming sources may only require none or minimal forms of processing such as filtering of irrelevant data or sampling of parts of the data. As for traditional forms of pre-processing in batches, there is a similar set of tools and methods available for Big Data tools as for ETL in DWH and RDBMS. The sub-tasks for *Data Processing* can include Data Cleansing, Data Integration, and Data Loading, albeit different (and a plethora of) Big Data tools are available for these tasks. For instance, custom MapReduce jobs can be written to pre-aggregate raw data in an HDFS into a form that enables detailed analytics later on. While there are also variants of ETL (e. g., ELT or Extract-Transform-Load-Transform (ETLT)), in Big Data scenarios the set and order of methods to be applied is completely flexible and depends highly on the case. For exploratory scenarios, where low-value data is inspected for valuable insights, heavy pre-processing should be avoided. In that case, source data could be loaded "as-is" into an HDFS and explored there. When dealing with potentially higher-value Big Data, e. g., logs from machines, where fault-patterns should be detected, data has to be cleansed and brought into a common form so that statistical algorithms do not work on faulty data such as missing or erroneous values.

Data Analysis. Naturally, the Data Analytics phase is concerned with conducting the main part of value generation. For Big Data, the set of traditional analysis methods is extended by several new ones, which work on semi-structured or unstructured data. For instance, sentiment analysis as a form of

text analytics became especially prominent with the widespread availability of semi-structured user posts from social networks, notably Facebook and Twitter (cf. Section 2.3.4). Also video and picture recognition, e. g., picture classification, saw many innovations due to the Big Data trend. For example, Microsoft uses a cloud-based machine learning classifier to detect faces of persons on a photo and guess their age (cf. [264, 263]). Also, analytics of structured data is possible with Big Data solutions. These can be NoSQL databases that offer a SQL-like query syntax, or NewSQL DBMSs as distributed form of traditional RDBMS, which supports horizontal scaling (cf. Section 2.2.2 and Section 2.2.7). Analytics can be eventually broken down to the basic building block that are various analytics methods, which are clarified upon in Section 2.3.3.

Data Usage. The final part of value generation is actually making use of data. This is typically the data, which has been analyzed in the step before, but also could involve (pre-)processed data for display. "Usage" includes several possible activities, some of which were explicated by the presented value chains. For instance, decision making is one possible and typical usage of such data. However, what kind of action is specifically taken with or after using this data, is out of scope at this point, as there are numerous possibilities, which are meant to be generalized with the term "usage". An implicit precondition for usage is that data is made accessible for any party using, be it a human or another machine. However, only through usage can data help to generate additional business value. This entails simple physical access or more complex visualization of it. However, visualizations are usually dedicated to human end-users, while access interfaces with various mechanics (e. g., push vs. pull).

Data Storage. Data Storage is meant as support infrastructure to persist any data along the value chain, when needed. The storage infrastructure takes the necessary form to fulfill these demands. For Big Data scenarios, a wide-range of novel options is available, HDFS and a plethora of NoSQL data stores being prominent choices. These come on top of distributed RDBMS for structured storage of "big" data volumes (cf. Section 2.2.7). The Data Storage activity

in the value chain has bi-directional interactions with the Data Acquisition and Processing as well as with the Data Analysis and the Data Usage phase. For the first one, the storage infrastructure can either load data directly after it has been acquired. This is data in raw form as promoted by the Data Lake pattern using HDFS (cf. Section 2.2.3). Furthermore, data can be stored after it as been pre- or post-processed, e. g., after data cleansing or filtering (i. e., pre-processing) or further data integration after data analysis or even data usage (i. e., post-processing). Thus, the storage is dynamically employed at the beginning, in-between, or after the Acquisition and Processing phase.

3.3 Data Warehouse Architecture Advancements

Although the DWH reference architecture illustrated in Figure 2.4 is a widely accepted template for building Data Warehouses, several advancements to it were proposed since its inception. In addition to that, the underlying technologies and tools, besides the outlined advancements in Section 2.2.7, have also undergone several changes and improvements.

As modern RDBMSs have evolved to cope with Big Data (cf. Section 2.2.7), so have Data Warehouses. In fact, even before Big Data emerged, these have adopted horizontal scaling approaches using shared-nothing architectures (cf. [DMS06, SAD+10, Cat11]).

DeWitt et al. note that such modern Data Warehouses allow for a horizontal partitioning of data, where no data (i. e., disks) are shared between nodes, respectively processors [DMS06, p. 3]. They contrast these with less capable *shared-memory* (e. g., MySQL and PostgreSQL) and *shared-disk* approaches (e. g., SAP IQ[2]). Warehouses, which support shared-nothing

[2] Previously called Sybase IQ [MF04]; https://www.sap.com/products/sybase-iq-big-data-management.html.

architectures include *Teradata*[3], *Netezza*[4], *Greenplum*[5], and *Vertica*[6]. SAP HANA[7] similarly features enhanced scalability, but is an IMDB at core, i. e., it processes data in-memory [FCP+12, FML+12]. Moreover, HANA and also SAP IQ feature *columnar storage* (cf. [MF04, FML+12]). Instead of organizing data per tuple, it is split up so that attribute values are stored conjointly and are materialized into their original tuples on demand. Grouping similar attributes yields several performance benefits, in particular for read-intensive OLAP queries, as, e. g., aggregations can work on co-located and possible even compressed data directly (cf. [Aba08, SAB+05]).

Nevertheless, besides these technological improvements, several conceptual advancements of Data Warehouses have been proposed. The following sub-sections provide a brief overview. In Section 3.3.1, a proposal to optimize the organization inside a Warehouse by Inmon is described. Next, possible augmentations of a Data Warehouse with Big Data are outlined in Section 3.3.2.

3.3.1 Data Warehouse 2.0 by Inmon

In [ISN08], Inmon describes a Data Warehouse *2.0* as an evolution from earlier Warehouse systems [ISN08, p. 9]. Drivers for this were not just technological advancements and prevalence of "online" processing due to widespread usage of networks and the Internet, but also the inclusion of unstructured, textual data [ISN08, p. 10]. Consistent with the Big Data trend, Inmon concludes that this data contains a "wealth of information" [ISN08, p. 10]. Moreover, he notes that both capacity- and performance-wise new advancements were made, which need to be exploited. Instead of relying on a single technology, a modern environment builds on several ones of them [ISN08, p. 10]. However, no specific technology besides Data Warehouses in general are mentioned. From a high-level perspective, there are no fundamental

[3] https://www.teradata.com.
[4] https://www.ibm.com/analytics/netezza.
[5] https://greenplum.org.
[6] https://www.vertica.com.
[7] http://hana.sap.com.

changes in the data flow. Data is ingested from various sources, which now particularly include textual data (e. g., blog entries or social media posts) and textual analytics as use case. What is modified is the Warehouse structure to accommodate this new kind of data.

In general, Inmon's Data Warehouse 2.0 reference architecture [ISN08] adds the following notable architectural aspects:

Organization by access probability. Instead of having one central layer for data access, there are four stages ("sectors") of access: interactive (second-span latency), integrated (24 hours to one month), near line (up to three to four years), and archival (after five or ten years) [ISN08, pp. 27f.]. Data is moved from the first stage subsequently to the archive if it accessed less often and gets older. Consequently, data volumes stored in this stage increase with each stage. For instance, in the interactive stage there is only a thin storage layer — if any — to allow for low latencies. The archive stage hosts majority of the data after some time.

Separate ETL for unstructured data. Text analytics becomes the second cornerstone besides structured (transaction) data analysis. In parallel to an existing data access and ETL layer, there should be tailored ETL process for textual data. As the regular ETL, they are partitioned into four access categories.

Speed-oriented access to transaction data. For low-latency purposes on the "interactive" layer, integrated transaction data can be queried with no or minimal ETL involvement.

In contrary to other approaches like HDFS and data lakes, unstructured data undergoes several phases of processing to be eventually stored in a relational database. Thus, a structured format is still the end goal of this proposal. Although being envisioned as independent from specific tools and technologies, an integrated DWH is still at its core.

3.3.2 Data Warehouses and Big Data

There are proposals, which aim to combine a Data Warehouse with Big Data and other novel technologies, e. g., in [133] [ZED+12, p. 23], [HPS+16, p. 26].

Here, three variants proposed by IBM [133] as representatives are briefly outlined to illustrate possible augmentations of DWHs with Big Data.

IBM sees three usage patterns of Big Data technology extending functionality of a Data Warehouse [133, 106, 360, 226]. These can be related to previously discussed technologies and patterns of Big Data Technologies (cf. Section 2.2). An integration of all three patterns into already introduced DWH reference architecture is illustrated in Figure 3.9.

Big Data Acquisition as Data Warehouse Source [106] [JSNJ15]. This pattern relies on a distributed Big Data acquisition and storage to collect data centrally, before all or parts of it are integrated into a Data Warehouse. Big Data processing in such a "landing area" can also provide certain preprocessing tasks for a Warehouse. Further, IBM sees this acquisition layer also as direct access path for real-time analytics, where latency-inducing processing layers can be omitted, as well as data provisioning for application level compute operations, which combine both DWH and data from central storage. This central storage facility resembles a data lake pattern, where data is collected centrally, even if it is not used immediately. Notably, this may necessitate a change to the acquisition and ingestion process of an ETL process, as operational systems are only indirectly accessed. A Big Data capable storage, e. g. HDFS or Apache Cassandra, could receive data to make it available to a Warehouse's ETL process [106]. This pattern is also promoted by [JSNJ15].

Big Data Storage for Archival [360]. Shweta and Nandi suggest that distributed Big Data tools can be used for improved archival mechanisms for Big Data. They state that archival of very old Warehouse data on tertiary storage prohibits such data from being used further for decision making. Their proposal features an "active" archive using distributed storage (e. g., HDFS) and compute (e. g., MapReduce) and query layers (e. g., Pig Latin) on top of "cold" archive data. That way, otherwise inaccessible data could be preserved for decision making. Notably, this approach is strictly for cold data only. It is not further explicated how to use both Data Warehouse and such

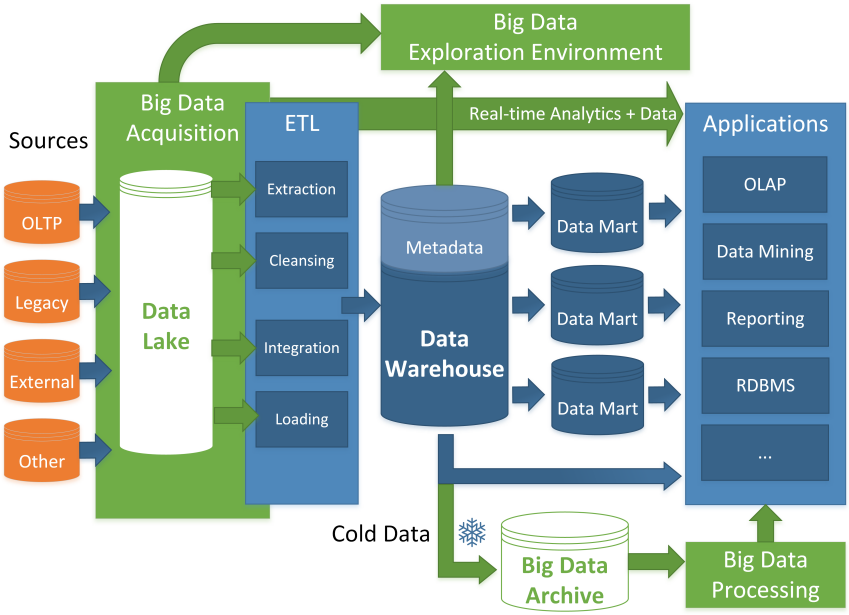

Figure 3.9: Data Warehouse Reference Architecture with Big Data extensions as envisioned by Shweta and Nandi. Source: based on [106, 360, 226].

archival data in combination (e. g., joining these data for analysis), which would be needed if hot data were to exceed Warehouse capacities. [360]

Big Data Storage and Processing for Explorative Analyses [226]. Just like the exploration warehouse concept in the realm of data mining (cf. Section 2.1.2), Khupat and Ahire promote a separate Big Data tool based system for exploratory analyses. Here, integrated data from a DWH is brought together with unprocessed or less processed data from a distributed Big Data system in a separate exploratory environment, which makes use various Big Data analysis tools, e. g., machine learning or stream processing. [226]

Notably, with all of these three patterns, Data Warehouse operations themselves remain as they are. There is no profound conceptual change inside a Data Warehouse architecture, only outside of its immediate system boundaries (i. e., before data comes in and after it leaves the system).

Apart from the aforementioned, a more profound alteration is proposed by Thiele et al. [TLH11]. While no specific Big Data technology is part of their **Data Warehouse 3.0** model, it is influenced by in-memory processing and stream analytics. In short, depending on data velocity (streaming data, micro-batches or streaming data) and by using either pull or push bashed mechanisms, layers of typical ETL pre-processing can be circumvented to establish a more immediate connection between BI application layer and data. Naturally, foregoing pre-processing means that more raw data is consumed. Notably, a DWH is still a core element of this model as target for integrated data, although use of technologies such as MapReduce for, e. g., ETL is not explicitly excluded.

These augmentations with Big Data illustrate the feasibility to combine traditional and novel technologies for decision-making. Additionally, more sophisticated architectural patterns emerge due to this technological combination. The original DWH reference architecture features a sequential pattern, where data flows from ingestion and pre-processing into the core Warehouse for storage. From there it is either more deeply analyzed, e. g., using OLAP analyses or the integrated data is disseminated in the organization (cf. "information" vs "analytics" processing in [HKP11, p. 153]). The combinations of Big Data render this flow more dynamic with several possible "paths" and activity alterations.

3.4 Big Data Reference Architectures

A selection of illustrative Big Data reference architectures found in a literature search are described and analyzed in the following. First, the respective reference architecture is elaborated upon and illustrated. The search process is documented in Appendix A. The particular selection of six examples has the goal to outline several facets of reference architectures, which include

their relationships to data value chains and the architectural patterns they inhibit. This provides motivation and background for this work's contribution.

The discussed Big Data reference architectures are:

1. Big Data Reference Architecture by Pääkkönen and Pakkala in Section 3.4.1.
2. NIST Big Data Reference Architecture in Section 3.4.2.
3. Big Data Solution Reference Architecture by Geerdink in Section 3.4.2.
4. Oracle Big Data Platform in Section 3.4.4.
5. Lambda Architecture by Marz and Warren in Section 3.4.5.
6. SOLID Architecture by Martínez-Prieto et al. in Section 3.4.6.

Finally, the section concludes with a comparative analysis of the examined Big Data reference architectures in Section 3.4.7.

3.4.1 Big Data Reference Architecture by Pääkkönen and Pakkala

The paper by Pääkkönen and Pakkala contributes two artifacts. Firstly, they construct a Big Data reference architecture model (cf. Figure 3.10). Secondly, they propose a visual classification of Big Data technology, products, and services into functional clusters, i. e., a technology map.

Pääkkönen and Pakkala use an inductive approach to construct the reference architecture, inferring from seven practical use cases. The findings from these cases are used as ingredients during reference architecture construction [PP15, pp. 2;19]. However, it is recognized that a conceptual integration of these various Big Data solutions into one "coherent" reference architecture has limitations. The reference architecture is intended to be technology-agnostic [PP15, p. 1].

Contribution Description

Their reference architecture depicted in Figure 3.10 has several high-level building blocks structured along typical value chain activities, such as "Data sources" or "Data analysis". Inside these blocks are functionality (e. g., "Extraction" in "Data processing") and storage elements, which are distinctly visualized. The reference architecture connects all building blocks to a value chain starting from "Data sources" to "Interfacing and visualization", except "Data storage" and "Job and model specification", which are foundation blocks. "Data storage" especially overlays all blocks from "Data extraction" to "Data loading and transformation". Their pipeline should represent a flow of data between the macro building blocks from "Data sources" to "Interface and Visualization". However, any concrete architecture based on this reference architecture can use arbitrary flows between these blocks to implement various architectural patterns as demonstrated in [PP15, Fig. 3 on p. 5].

Data sources are the inception point of the Big Data reference architecture. Abstract data sources are characterized by velocity ("mobility") and variety ("structure") (cf. Section 2.2). velocity distinguishes streaming data from resting data. Their variety dimension comprises the three typical characterizations, namely structured, semi-structured, and unstructured data. Interestingly, data or problem size is rarely considered in the reference architecture model (cf. Figure 3.10).

Data extraction is the first part of this model's three-part data ingestion routine, which resembles typical ETL steps (cf. Section 2.1.2). Pääkkönen and Pakkala divide the extraction task according to data velocity, resulting in one extraction task for regular and one for streaming data. For both, there is an optional temporary storage option in the "Data storage" block. However, extraction is aimed primarily at resting data, whereas especially for streaming it is seen as optional [PP15, pp. 3ff.].

Data loading and pre-processing is envisioned by the authors to contain a raw data store as optional storage. It should receive unprocessed or uncompressed data, which can be cleansed, processed and transferred into the

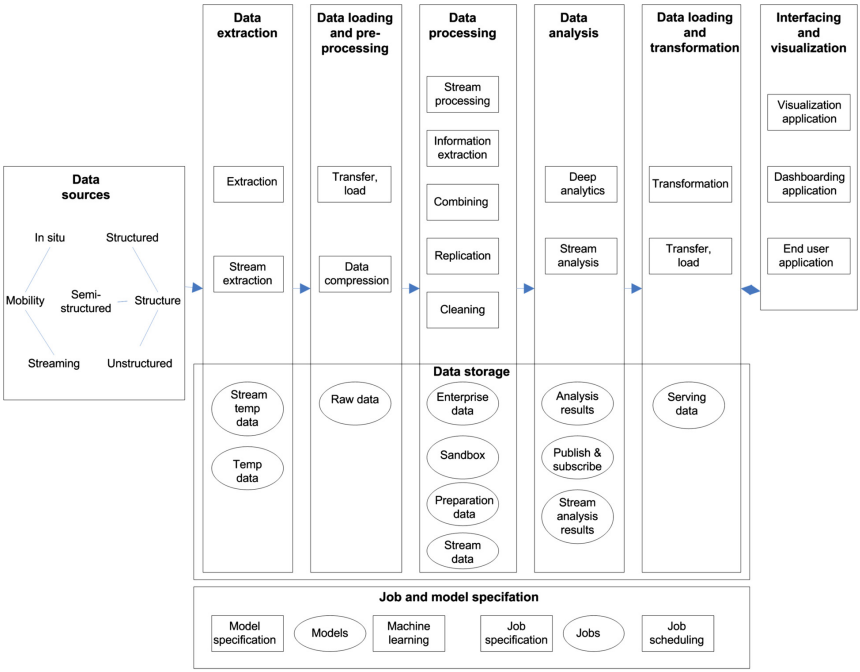

Figure 3.10: High-level view on Big Data reference architecture by Pääkkönen and Pakkala, structured along value chain activities, with data stores as ellipses, functionality as rectangles and data flow as arrows. Source: [PP15, p. 3].

next block afterwards. This is comparable to data lake approaches. It is also seen as an alternative option for the temporary storage from the previous block. Besides the physical data loading and transferring functionality, the authors see data compression as possible precursory task to reduce data volume [PP15, p. 3].

Data processing encompasses several possible activities. For instance, there are the typical ETL-related tasks as data cleansing and integration ("cleaning" and "combining") for data from the raw data store. Extraction of structured data ("Information extraction") prepares data for "deep analytics". Lastly, processing of streaming data ("Stream processing") and technical replication of data are considered. For storage options in this block, the authors present four alternatives. For instance, a "enterprise data store" holds high-value data for productive use, which has been enriched through cleansing and other forms of processing. It is not limited to structured data, but can also use, e. g., Hadoop or HDFS as physical storage [PP15, p. 6]. A "sandbox store" is a typical, separate exploratory environment [PP15, p. 4], while a preparation data store can be employed to save cleansed and processed data temporarily. Again, streaming data is considered separately with an optional "streaming data store". [PP15, pp. 3f.]

Data analysis consists of two activities, "Deep analytics" and "stream analysis". The latter deals specifically with analyzing streaming data (e. g., cf. Section 2.2.6). The former is equated with the "execution of batch-processing jobs" for resting data. Results can either be stored in the original location or routed to an "analysis results store". Similarly, there is a "stream analysis results store". Lastly, a "publish & subscribe store" is used for data flow management (cf. Section 2.2.6).

Data loading and transformation is tasked to optionally "transform" data into formats that can be used by other applications down the line. Transformed data can be stored in a "serving data store", which offer the transformed data to the consuming applications. A typical example for this usage pattern is OLAP, where multidimensional models are created from integrated source and are stored separately (cf. data marts in a DWH, Section 2.1.2).

Interfacing and visualization encompasses three generic application types for this task, namely "visualzing", "dashboarding", and "end user" applications. For instance, "visualizing application" refers to complex BI apps or simpler dashboard applications for displaying of KPIs. The last archetype, "end user" mobile apps, are targeted at end users, i. e., the customers of the involved use case companies (cf. [PP15, p. 8]), characterized by limited customizations abilities. [PP15, p. 4]

Data storage is the foundation for the five building blocks regarding extraction, pre-processing, processing, analysis, and transformation. It comprises mostly optional or alternatives for storage options. Importantly, the options are not tied to specific technologies but rather to functions they should execute. These functions are based on the observation on the seven source use cases.

Job and model specification is concerned with controlling training and execution of the other building blocks. A "job" is an inherent concept of batch processing solutions like MapReduce (cf. Section 2.2.5). With YARN, the Hadoop ecosystems offers a dedicated tool for job management (scheduling and execution), exactly as depicted by Pääkkönen and Pakkala. Additionally, the authors have included the specification and management of machine learning models (cf. Section 2.3) as supporting activity.

The *second contribution* is a classification map of Big Data technologies, products, and services [PP15, p. 12]. The authors assign instances of these three types into seven different macro groups: Data collection & storage, Data analysis, Benchmarking, Novel technology frameworks, Virtualization, Cloud-based solutions, and Commercial products and services. Primarily, the chosen technologies, products, and services are based on the seven use cases. However, these are also augmented with ones found in selected related work (e. g., "Voldemort" in a "Key-value stores" sub-group) [PP15, p. 12]. The names of the sub-groups are based on the products in the group (e. g., document stores and graph databases).

Evaluation

Overall, the Big Data reference architecture by Pääkkönen and Pakkala has several structural similarities to the previously proposed Big Data value chains and to the synthesized universal value chain as well (cf. Section 3.2.2), as it depicts building blocks that resemble high-level functionalities, which can be found in the unified value chain such as data processing. Several elements of this reference architecture are considered optional. Possible combinations of, e. g., tasks and stores, are described in the text only. Technologies are also not part of the reference architecture, but are separately discussed and have no explicit connection to the Big Data reference architecture.

The authors employ a strictly inductive approach for their reference architecture. It is based on seven use cases, which are published by the companies that implemented the respective solutions and are, thus, publicly available. This means that the "super-set" (cf. [PP15, p. 19]) of all solutions in the use cases strongly influences the contribution. The construction of the reference architecture is preceded by a thorough analysis of the Big Data solutions employed by the selected companies. This allows to gain detailed empirical insight into the implementations as they have been described by the implementers (e. g., Facebook and Twitter [PP15]).

Naturally, a strictly inductive approach has limitations, which inhibit the generalizability of the reference architecture. The authors do not argue how using these seven use cases results in a generally applicable Big Data reference architecture. While the use case selection is appropriate (e. g., Facebook, Twitter) the question remains if these are sufficient to create a truly universal Big Data reference architecture. For instance, it remains unclear whether the various data stores, which became part of the model, are comprehensive in the field of Big Data.

3.4.2 NIST Big Data Reference Architecture

The NIST Big Data Reference Architecture (NBDRA) is published by the NIST Big Data Public Working Group, which consisting of industry, academia, and government actors from the United States [NIS15b, p. 1]. The NBDRA is

a conceptual reference architecture, which aims to be independent of specific technologies as well as infrastructure components and vendors [NIS15b, p. vii]. The formulation of the architecture is primarily based on a literature survey focused on white papers handed in by organizations [NIS15a], where the aforementioned proposed Big Data reference architectures. These publications are part of the *NIST Big Data Interoperability Framework*, which consists of seven volumes with dedicated topics. Apart from the previously mentioned two volumes, there are other volumes that, e. g., deal with Big Data taxonomies and definitions [NIS15b, p. vii].

Their work is also motivated by an understanding that there is no consensus on fundamental concepts of Big Data yet such as the essential characteristics and patterns of Big Data environments. These include a reference architecture as well [NIS15b, p. 1]. Conclusively, this Big Data reference architecture serves the purpose of standardization instead of facilitation.

Contribution Description

The NBDRA in cf. Figure 3.11 is actually fueled by two different value chains. Firstly, it organizes five functional components, which should constitute technical roles a Big Data system. Generally, these can be fulfilled by humans or by machines, or both [NIS15b, p. 13]. For the components, typical tasks therein as well as data flows in a value chain are illustrated (cf. Section 3.2). This outlines the layers, which refine "raw" data from data providers into valuable information which can be accessed or visualized by other consumers. On the other hand, it is vertically organized by according to an "IT value chain". Here, the layers of IT services backing the value chain are exposed, starting from infrastructure components as foundations to processing capabilities, which can be consumed by the "application provider" for tasks along the information value chain. [NIS15b, p. 11]

The five functional components are as follows [NIS15b]:

Data Providers are considered a generalization of data sources as a service, which "introduce[s] new data or information feeds" into a Big Data system. Providers can both be data sources in a traditional sense, such as other (Big

Data) systems, databases, or legacy sources, as well as data generators such as sensors, external data vendors or even humans that input data manually. Data is offered to the *Big Data Application Provider* through a defined interface or service. Interfaces are not just determined by the variety of the source, but other Big Data characteristics (e. g., volume or velocity) and system requirements as well. When data is handed over, it can happen via push- or pull-based mechanisms and be conducted synchronously or asynchronously. [NIS15b, pp. 13f.]

The **System Orchestrator** is the designated role for managing and orchestrating a Big Data environment. It is a set of both human and IT resources to fulfill this task. One responsibility is the integration of "data application activities into an operational vertical system" [NIS15b, p. 13]. Another function is to manage architectural components so that workloads can be executed on the overall architecture. For instance, Apache YARN provides such workload management for Hadoop environments. [NIS15b, p. 13]

The **Big Data Application Provider** works along a typical value chain ("data life cycle" [NIS15b, p. 15]) depicted in the architecture. The value chain consists of "Collection", "Preparation", "Analytics", "Visualization", and "Access" [NIS15b, p. 15] (cf. Section 3.2). For instance, the collection activity is seen as interface point to the Big Data Application Provider role with a data provider. "Collection" and "Preparation" activities are meant to include typical ETL tasks (Extraction and Transformation/Loading respectively), especially in the Preparation step. Here, it is explicitly avoided to mention specific applications or technologies for these activities. The "Analytics" activity also features an abstract description and is structured on a typical spectrum of batch and streaming solutions categories. The usage of data (cf. Section 3.2) then is split into "Visualization" and "Access" activities. "Visualization" encompasses the "presentation to the Data Consumer". Hereby, dedicated tools, own implementations, or specialized frameworks for that matter can be leveraged. On the other hand, the "Access" activity is concerned with rendering analyzed or visualized data accessible to a data consumer (machine or person).

Big Data Framework Providers are responsible for components along the aforementioned "IT value chain", which consists of infrastructure, (data) platforms, and processing (generic and analytic), supported by communication and resource management. These components are layered, meaning that infrastructure is on the bottom and provides the most basic services to data platforms, which in turn service processing components. [NIS15b, p. 17]

> **Infrastructure Frameworks** are responsible for both hardware and software infrastructure, which support the actual processing or data storage platforms. These include networking as well as computing (CPUs and other processing units such as GPUs and memory, i. e., RAM) and storage solutions (e. g., SAN or Network Attached Storage (NAS)), both physical and virtualized (e. g., Software Defined Networks (SDN) or Virtual Machines (VMs) respectively container-based virtualization solutions such as Docker[8]). Also, external infrastructure in form of power supply, cooling, and security must accommodate a Big Data solution. This infrastructure can be local or cloud-based, where a service provider is responsible for maintenance and supporting the basic infrastructure. All these facilities should be planned to support required scaling for Big Data solutions to be deployed there.
>
> **Data Platform Frameworks** deal with logically organizing and distributing data. The authors summarize typical types of logical organization and classify them into distinct categories (cf. Section 2.1.1 and Section 2.2). These are either *in-memory*, *file system* organizations with the dimensions distribution (centralized/distributed) and form (delimited, fixed-length, binary), or *indexed* organizations. The latter subsume structured, relational storage forms as well as typical semi-structured forms ("key value", "columnar", "document", "graph") [NIS15b, pp. 20f.]. However, no specific technologies such as RDBMS or NoSQL data stores for indexed storage or HDFS for distributed file storage are mentioned [NIS15b, p. 21]. It is stressed that the data organization approach depends on the characteristics of problem to be solved.

[8] https://www.docker.com.

For velocity challenges and time-critical analyses, in-memory solutions that rely on high amounts of main memory are recommended [NIS15b, p. 22]. For throughput challenges with respect to volume and velocity dimension occur combined, distributed file systems are recommended. Depending on data variety challenges, indexed approaches can be used, as these allow to quickly find and access data sets, address volume and velocity challenges as well, because indexing may allow to access data more quickly than scanning a file system [NIS15b, pp. 23f.]. The specific form of indexed storage should be selected according to "data structure complexity", "access flexibility", and "data linkage complexity" (cf. Fig. 4 in [NIS15b, p. 24]). For instance, key-value stores offer only a simple storage structure and linkage options to other data. On the other hand, graph databases allow to store complex graphs and particularly focus linkage by their inherent description of relationships. Document stores are less complex, and have lower linkage complexity, but have more flexibility in accessing data, since documents are easily accessible and navigable [NIS15b, p. 24].

Processing Frameworks provide the foundation to actual application that fulfill Big Data application requirements and can deal with Big Data characteristics at hand (Vs; cf. Section 2.2.1). These determine how analysis and processing functionalities are organized. The authors confirm that a plethora of platforms and technologies is available for this [NIS15b, p. 25]. They also note that frameworks may offer different bandwidths on the spectrum between *batch* and *stream processing*. It is further pointed out that *interactive* access during data retrieval would become more prevalent [NIS15b, p. 26]. Functionality of these frameworks is grouped along three sequential activities ("processing phases") in the value chain from "Data" to "Information/Results" ("information flow"): data ingestion, data analysis, and data dissemination [NIS15b, p. 25]. This value chain is distinct from the overall value chain and regards information flow for the specific subtask of a framework. Apart from that, the "primary location" of processing frameworks is mapped to three processing activities. Stream processing frameworks are seen

> to occupy the whole spectrum from "ingestion" to "dissemination". This is due to the needed timeliness implied by using a streaming solution. Batch processing resides mainly in analytics, but with slight connection to the ingestion and dissemination phase. Interactive accesses are less involved at the beginning of the information flow, but have a stronger focus in the dissemination phase (cf. [NIS15b, p. 25]). Notably, data transformation could take place during any point of the three value chain activities, depending on use case requirements. For instance, a timely analysis might need an incremental analysis. The categorizations made herein should not be perceived as distinct and tools and frameworks may fall into more than just one of these categorizations [NIS15b, p. 26].

Data Consumers mirror the role of the Data Providers. Here, consumers make use of data accessible to them. Again, they can be either humans or machines. Usage activities should include typical application tasks such as reporting and visualization (i. e., BI applications), searching, retrieving respectively downloading data. Interaction of consumers happens with an application provider, through defined service interfaces (visual and non-visual ones). Interaction patterns not just include traditional BI-style or batch-based ones, but also novel stream- or push-based consumption modes (e. g., as offered by tools like Apache Zeppelin[9]).

The scope of the NBDRA also includes several support aspects beyond any core functionality on the value chain, such as management, security and privacy, or technical management capabilities ("Resource Management").

Evaluation

The NBDRA demonstrates the wide spectrum of scope and abstraction of references architectures. This reference architecture covers a wide range of Big Data usage in an organization. There is no restriction regarding possible problem subdomains. Not just aspects alongside a value chain are described,

[9] https://zeppelin.apache.org.

but also supporting elements on the periphery (e. g., data providers). These include organizational (e. g., roles of employees) alongside technical aspects. The NBDRA also discusses and assigns several roles and responsibilities (e. g., system orchestrator), instead of discussing specific solutions or parts of them. For each of these roles, it is attempted to outline the spectrum of possible solutions. The abstraction role descriptions (e. g., consumers that can be machines or humans) with the wide range of possible responsibilities and tasks for them underline the holistic approach to depicting Big Data use in an organization. As one of its goals is standardization, related elements alongside data processing in an information value chain are depicted in an abstract fashion. Moreover, not just areas, which are directly covered by usual Big Data technology (e. g., HDFS or MapReduce) are covered, but basic support infrastructure (e. g., cloud vs. local deployment or operating systems) are discussed as well. In that sense, this reference architecture is closer to a reference model than other Big Data reference architectures.

The approach is fueled by several cases handed in the working group's stakeholders, but technology specifics are abstracted. Technology is categorized functionally to outline the whole solution spectrum, while remaining independent from single technologies (e. g., the different categories for data organization). Detailed architectural usage patterns are not directly discernible from the visual representation of the reference architecture. In particular, specific products or tools are rarely mentioned. For instance, neither a DWH nor data lake usage pattern can be easily read from the model, although the reference architecture does not inhibit deducting a custom architectural pattern.

In summary, the different focus of the NBDRA makes it more useful to view Big Data usage holistically and survey the whole spectrum of possible solutions or parts of them. As its goal is standardization and not facilitation, it is more challenging to use such a reference architecture as basis to build a concrete architecture. Nevertheless, it could be used for other purposes, such as validating solutions respectively architectures against this. It could be used to verify whether all necessary elements in a solution are considered. Additionally, the classifications of technologies could provide useful information when constructing a customized BI architecture.

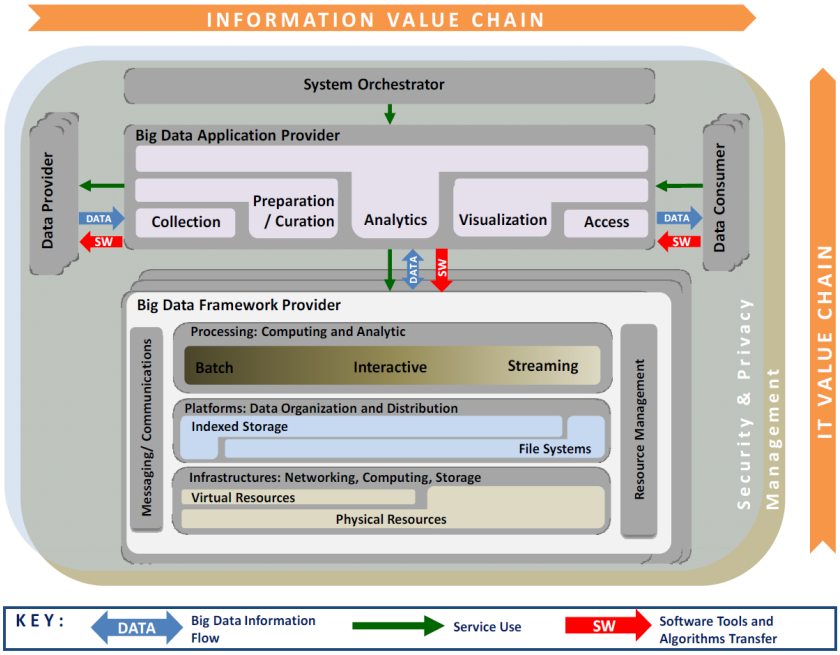

Figure 3.11: NIST Big Data Reference Architecture. It shows functional components in the overall system landscape, depicts usage and provision of services as well as aligns it along an information and IT value chain, while being interwoven with management and security approaches. Source: [NIS15b].

3.4.3 Big Data Solution Reference Architecture by Geerdink

Geerdink proposes a Big Data *solution* reference architecture. It is described as a "skeleton for a solution, where the elements are templates or outlines for components", which contains hardware, components, patterns, and principles [Gee13, p. 72]. The reference architecture is designed as an Information Systems artifact and deductively built based on a design science method by Hevner [HMPR04] (cf. Section 1.2). For construction, "business requirements" and a literature analysis are employed, which are not detailed in the paper [Gee13, p. 72]. It is designed to be a "best-practice reference architecture", which is rooted in practical experience of experts [Gee13, p. 72].

For building the Big Data reference architecture, it is stressed that not just dedicated Big Data reference architectures are taken into account, but practical aspects are as well, namely "hardware and software components", "architecture principles", and "best practices". This is in line with contents of reference architectures (elements, relationships, and architectural patterns). This reference architecture is designed to provide guidance to implementers (i. e., facilitation) [Gee13, p. 72].

While the reference architecture is not targeted at a specific industry, the target audience is assumed to be larger organizations, which can afford a Big Data endeavor. The assumptions is that such an amount of resources (financial and human) is required to architect a Big Data project using the solution reference architecture that only organizations of the aforementioned size are suitable. [Gee13, p. 72]

The main goal of the entire solution is to "predict the future using large sets of enterprise data combined with open data sources" [Gee13, p. 72].

Contribution Description

The reference architecture is visually modeled using ArchiMate, an open modeling language for Enterprise Architectures (EAs) supported by a dedicated specification [378].

It consists of three layers (cf. Figure 3.12). These are a business, an application, and a technology layer as specified by ArchiMate. Through these layers, a flow of data and information is designed, which permeates the strictly layered architecture.

Business Layer. This layer contains a business process (data flow) for prediction purposes. The activities are ordered sequentially and range from data import, to processing, analysis, and decision making. It resembles a typical Big Data value chain (cf. Section 3.2).

Application Layer. Consisting of data and engines to process them, the application layer connects technology with the business layer. For each of the activities in the business layer, there is an equally named engine (e. g., "Import Engine" for "Import Data" in the business layer). However, for the decision activity, only a visualization layer is envisioned. Moreover, this layer has a dedicated management engine for coordination, which is connected to the importing, processing, and analytics engines. Lastly, there is a dedicated "Raw Data" element, which is used by the importing engine.

Technology Layer. Here, infrastructure technology for data import, processing, analytics, and decision making is located. These components are accompanied by data elements, which indicate various processing staging of data (e. g., "Imported Data" or "Processed Data"). The author sees internal "Enterprise Data" and external "Open Data" as the two basic data sources for Big Data solutions. These comprise "Raw Data". After this has been imported through an importing engine, it is stored in a file cluster, which is based on a distributed file system. For storage of processed data, distributed databases reside on a database cluster. Lastly, an analytics cluster houses "analytics databases". These "clusters" can reside on either on-premise or in the cloud (cf. Figure 3.12).

Evaluation

The *Big Data Solution Reference Architecture* adds various insights into Big Data reference architectures. It covers a typical Big Data value chain from

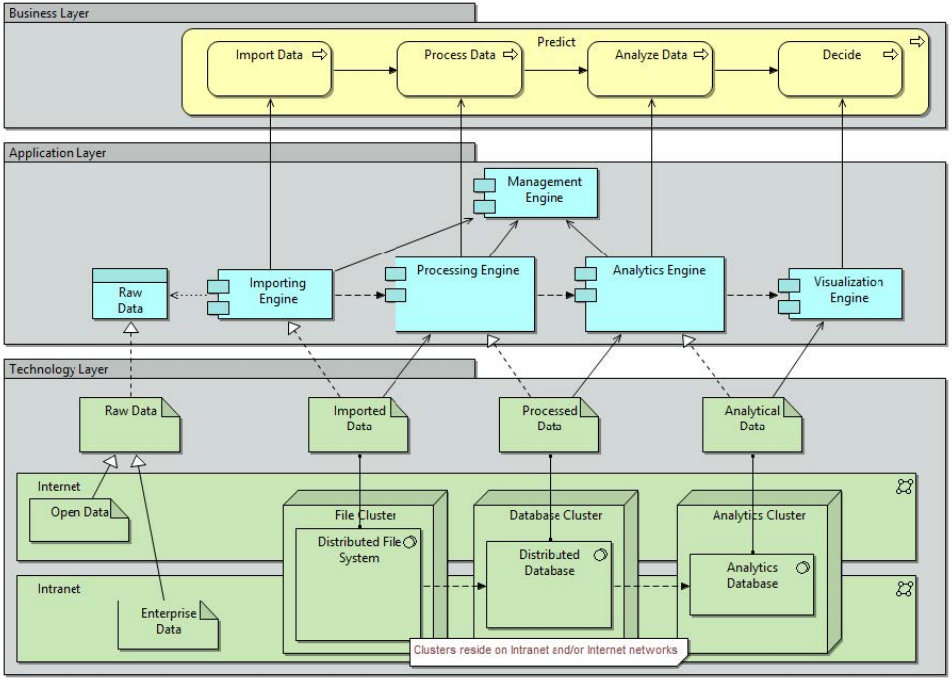

Figure 3.12: Big Data Solution Reference Architecture component and interfaces. Source: [Gee13].

ingestion to decision in an abstract fashion. It connects this with specific application types ("engines"), which alone or conjointly work on value chain activities, highlighting the relationships between value chain and application types. These are connected with data, computing resources and broad technology classes, namely distributed file systems, distributed databases, and analytics databases. One defining characteristic is that these systems should be (horizontally) scalable on clusters to accommodate Big Data. Regarding data, only internal enterprise data and external open data are meant to cover the 3Vs of Big Data (volume, variety, velocity). ArchiMate as standardized modeling language adds additional consistency and clarity to the representation. Although a business layer and consideration of agile development methods for the model represent some organizational aspects [Gee13, p. 75], the reference architecture focuses on technology. The model recommends usage of free and open source software.

Specific technology (e. g., NoSQL data stores) or tools (e. g., MongoDB) are not part of the reference architecture. However, technologies as components are documented as part of the research process outside the paper from the conference proceedings. Similarly, a list of selected free and open-source products is provided for all sub-tasks in the application layer of the reference architecture (e. g., for data transformation).

All elements are seen as optional to allow a customized creation of a specific solution architecture based on this model. For instance, storing ingested data can be left out for real-time analytics [Gee13, p. 74]. Nevertheless, the reference architecture offers no specific guidance on which of its elements to select in which case. This guidance would encompass selecting relevant elements from the reference architecture, identifying suitable sub-tasks in the application layer, and selecting appropriate and fitting technology and products. While the reference architecture should guide "facilitation", it is not an explicit goal to offer guidance on this selection processes.

[Gee13] details the generic development approach according to Angelov et al. and the visual representation of the reference architecture without going deeper into its elements (e. g., details of the engines in the application layer are not described). Moreover, it does not detail the specific process of

literature review, theoretic background, and survey questionnaire and the specific synthesis of these elements.

3.4.4 Oracle Big Data Platform

Oracle presents a "unified reference architecture" based on their previous experiences in deploying various Big Data tools for their customers [HPS+16, p. 2]. As a company, Oracle offers various commercial traditional database products like the Oracle database [315]. Additionally, several novel technologies are offered by them, e. g., an in-memory extension (Oracle TimesTen [316]) or a NoSQL data store. Besides that, larger appliances including special hardware with added services and cloud solutions are available (e. g., cf. [299]). For Big Data problems, they also advocate freely available Apache tools from the greater Hadoop ecosystem (cf. Section 2.2.5). Recently, Oracle focuses on cloud-based solutions, which can capture the whole business life cycle of information generation, preferably with own solutions (cf. [118, 327, 120]).

The white paper [HPS+16] describes some details of a reference architecture, but also a more general approach, including organizational aspects, to Big Data analytics. An earlier white paper, [Ora14], focuses on more technical details of the reference architecture discussed in the newer paper.

In general, Oracle stresses that its ultimate goal with its reference architecture is to "understand how these technologies relate" to status quo of data management and give guidance on "how best to combine them (if at all) into a coherent platform", which is suited for both management and facilitation of knowledge discoveries [Ora14, p. 1]. Specifically, [Ora14] sets out to describe *how* their reference architecture "can help to integrate [all] information [so] that [it] can be exploited for commercial gain." [Ora14]. [HPS+16] should serve as a foundation to Big Data capabilities, with products focusing on Oracle and open source products, and introduce its reference architecture and architectural choices. With that, an "approach and guidance" should be offered both conceptually and specifically by demonstrating three selected customer use cases [HPS+16, p. 2].

Contribution Description

Oracle terms its reference architecture "The Oracle Big Data Platform" (see Figure 3.13). It is based on architectural patterns that have emerged in various deployments the firm has conducted. Seven component blocks are aligned to show a flow of data through processing and analytics to its results, i. e. along a typical value chain [HPS+16, p. 16]. The process of how it was derived is not documented. These components only contain abstract, functional elements, e. g., a "Warehouse" or a "[Data] Reservoir" for "Data Management" without mentioning specific technologies. However, the seven components are put in relation to each other, indicated by data flow arrows between them. Nevertheless, there are no relationships between elements inside these components or more complex data flows. In spite of this, Oracle describes that the lower half of the model could be considered an "innovation" (i. e., exploratory) value chain to create new insights, while the top half would be for more standardized decision support (e. g., daily reports) [HPS+16, p. 18]. It also does not contain specific tasks, e. g., data pre-processing (which is also not explicated in this model).

Data only visually depicts various-structured data sources (e. g., database "transactions") or their typical origins (e. g., smartphones, "machine-generated" data). Explicated in their "capability map", all three Vs are covered (see Figure 3.14).

Fast Data covers stream processing, whereas "events" and actions made from stream processing are aligned alongside generic "streams". Oracle asserts that decision-making while stream processing requires a "decision context", which is especially supplied by regular data from a "Reservoir". [HPS+16, p. 17]

Data Management includes both unstructured and structured data management. A reservoir should include both storage and (parallel) processing for data, which is not strictly structured, e. g. Apache Hadoop or even a staging area for an ETL process [HPS+16, p. 17]. Mission-critical business data should be collected in a Data Warehouse or data marts.

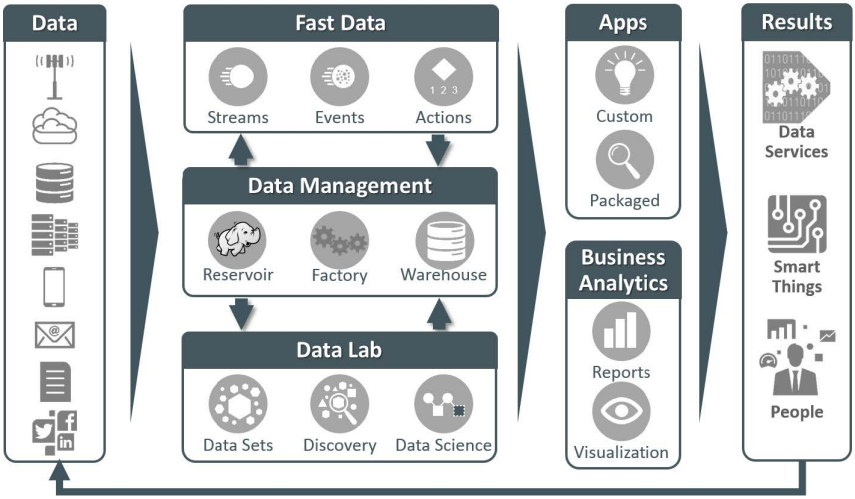

Figure 3.13: Oracle Big Data Platform is a conceptual, unified reference architecture based on the company's deployment experiences. Source: [HPS+16, p. 16].

Data Lab represents a separate environment for "discovery of new knowledge". It not just covers analytics, but also storage and processing functionalities. These should enable to quickly access data and analyze it with heavy statistical methods (cf. exploration warehouse concept and data mining in Section 3.3). This represents typical exploratory analyses patterns, where the business question is still unknown and new knowledge should be created, which is later incorporated into a stable value generation process (cf. Section 3.2).

Apps should contain programmatic access methods (i. e., APIs) and other adapters, which allow custom built or pre-made applications to access not just stored data, but also processing capabilities from "Fast Data", "Data Management", or the "Data Lab".

Business Analytics is separated from other applications and indicates BI tools, which cover traditional BI use cases such as dashboards and reports (cf. Section 2.1.2).

Results depicts three different kinds of sample output recipients for information generated by the reference architecture, namely services utilizing the data, users ("people") working with the data, as well as "Smart Things" (i. e., IoT and a wide range of other smart devices).

To fill these abstract roles, Oracle also provides a "capability map", which organizes abstract data categories, technologies, products, concepts, and applications alongside a value chain (see Figure 3.14). It is surrounded by supporting aspects, such as hardware and governance [HPS+16, p. 17]. It constitutes the second part of the contribution. It can be understood as ingredients, which can be filled into the abstract elements (e. g., "Reservoir") of the aforementioned reference architecture, but also roles for these are explicated (e. g., "Reservoir" is placed alongside the "Analyze" value chain activity). Here, specific technologies such as RDBMS or products such as Hadoop and MapReduce are introduced and allocated to value chain activities.

Further, sample products and tools for aforementioned technologies are listed. However, Oracle exclusively focuses on its own products and tools from the Apache Hadoop ecosystem.

The visual models are accompanied by textual descriptions, which elucidate some of the decision factors for certain components of the reference architecture.

In general, data processing capabilities should be aligned to characteristics of Big Data (i. e., the 5V) [HPS+16, p. 15]. Distributed parallel processing architectures are recommended for large data sets, but can also be tuned towards velocity [HPS+16, p. 15]. However, these might be separate pipelines. Specifically, *Lambda* (cf. Section 3.4.5) and *Kappa* architectures are mentioned as suitable in this context. For "rapid" evaluation of unstructured data (which is not real-time streaming data), Hadoop should be used for batch-based cases and Apache Spark for in-memory based applications [HPS+16, p. 17].

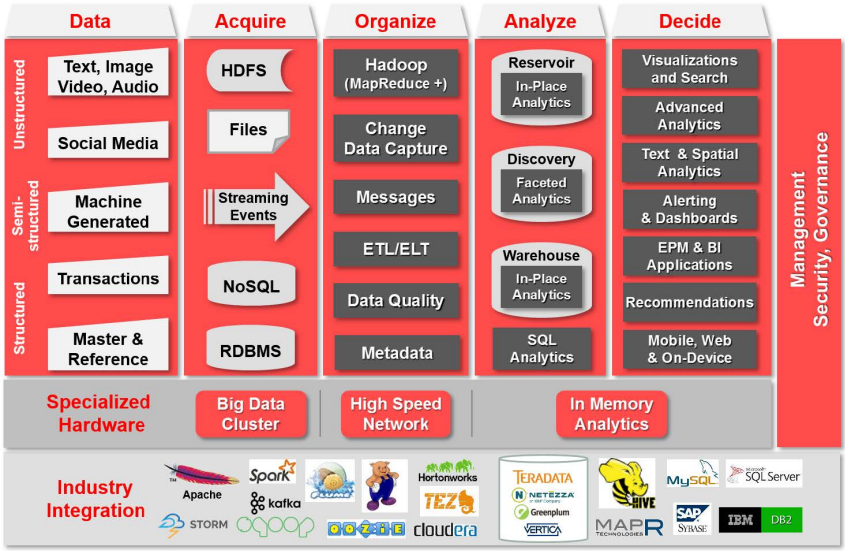

Figure 3.14: Data and Technology Map by Oracle, organized alongside a Big Data value chain. Source: [HPS+16, p. 17].

For real-time storage requirements, key-value data stores (i. e., NoSQL data stores), are recommended, due to their index-based, "high performance" retrieval [HPS+16, p. 14]. On the other hand, MapReduce is recommended for batch processing. The authors note that it would be common to move "MapReduced" data to a Data Warehouse or to another analytics environment. The reasoning is that existing BI capabilities could be leveraged [HPS+16, p. 15]. They also promote dedicated "discovery labs" or "sandboxes" (comparable to the concept of an exploration warehouse, cf. Section 2.1.2), which should only be kept as long as needed [HPS+16, p. 15].

Regarding enterprise usage of Hadoop, it is noted that it is advisable to integrate it with existing BI systems such as a Data Warehouse [HPS+16, p. 15].

Evaluation

The proposed reference architecture by Oracle includes several elements also found in academic or company-independent publications (e. g., as outlined in the sections before). Especially the capability map includes several known Big Data and traditional technologies such as RDBMSs, HDFS, NoSQL data stores, and MapReduce. Moreover, it illustrates usage patterns already exposed in literature, like a Warehouse or a dedicated exploration Warehouse or store, albeit with the inclusion of more recent technologies (e. g., scalable DWHs). Also, the usage pattern of a data "reservoir", especially using HDFS, is promoted and essentially resembles what is known as data lake (cf. Section 2.2.3).

The capability map in Figure 3.14 is a structured collection of tools, technologies, approaches, and data. While it shows the available (Oracle) solution space, it does not provide guidance on choosing its components. The conceptual reference architecture in Figure 3.13 also outlines a "total" picture of usage patterns in analytics solutions, in broad details and without specific technologies or tools. In general, the reference architecture depicts data flows through arrows so that arbitrary combinations of custom patterns are possible (cf. Figure 3.13). As before, which relevant subset of this solves a specific problem, is not outlined. Nevertheless, through supplementary texts, Oracle gives broad advice on technology classes to be used under which circumstances. In spite of this, neither of the two documents serves as explicit guide to build a specific architecture for a given problem. As the presented product solution space is focused on Oracle and open-source products only [HPS+16], the proposed reference architecture and guidance is not truly universal, although several widely used technologies and concepts are incorporated. In addition to this, the offered guidance is only unstructured and not inherent part of the proposed models.

3.4.5 Lambda Architecture by Marz and Warren

Marz and Warren propose their *Lambda Architecture* as "new paradigm" and approach for Big Data applications [MW15, p. 2]. A paradigm denotes a

pattern or model [321]. Thus, this contribution can be placed on an abstract spectrum of reference architectures, nearing architectural patterns.

The authors aim to address issues of size (i. e., scaling) and complexity of Big Data applications [MW15, p. 2]. According to them, traditional RDBMS technology inhibits scaling to Big Data, as a very complex approach involving horizontal partitioning and manual management, especially in cases of partition faults, would be needed [MW15, pp. 3ff.]. They assert that simpler data models and storage modes from Big Data solutions enable better scaling to cope with Big Data problems [MW15, p. 6]. However, fitting complex applications to a simpler model could become challenging [MW15, p. 7].

Their contribution aims to illustrate the basic principle that allows to build scalable Big Data systems. It is explicitly not bound to any technology, although the authors use some as examples [MW15, p. 2]. Specifically, systems built based on their Lambda Architecture should exhibit the following properties [MW15, pp. 7f.]: "robustness" (also in case of human errors), fault-tolerance, "low latency reads and updates" (for streaming applications), horizontal "scalability", "generalization" (i. e., being used as pattern or reference architecture), "extensibility", "ad hoc queries", "minimal maintenance", and "debuggability".

One guiding principle of the contribution that all queries for data are seen as a function over all data in a system. The Lambda Architecture should enable users to express any function on any (subset of) data [MW15, p. 7].

Contribution Description

The Lambda Architecture, depicted in Figure 3.15, is divided into three distinct layers [MW15, p. 14]:

Batch layer. This layer contains the so-called *master dataset*, which seen as "source of truth" from which the other layers rebuild their information repository (e. g., after failures) [Mar15, p. 27]. Thus, appropriate replication and backup mechanisms are crucial. Data in the master dataset should be stored in its "rawest" form, as raw data contains the highest granularity. The assumption is that more questions could be answered with raw data than

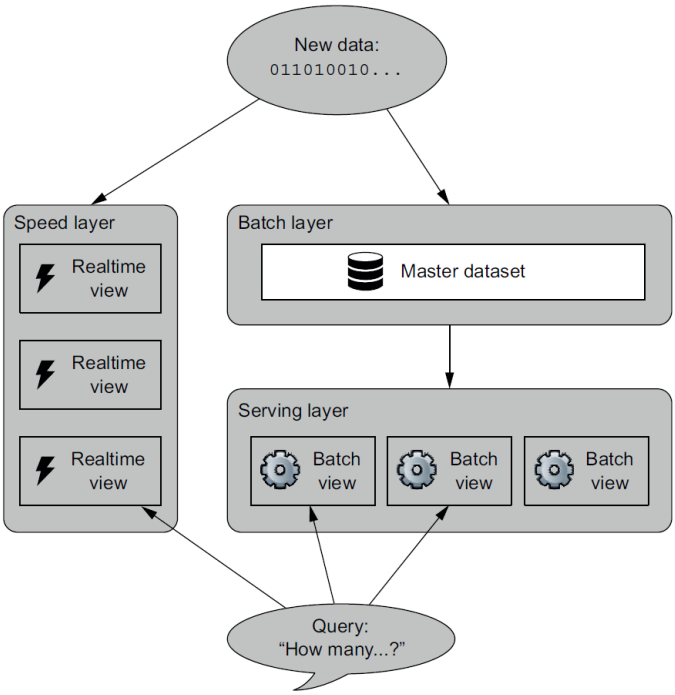

Figure 3.15: Lambda Architecture by Marz and Warren. Source: [MW15, p. 19].

with any other form of "derived" data (cf. [MW15, pp. 31f.]). Notably, derived data can be stored if the transformation functions are "simple and accurate" [MW15, p. 33], e. g., when extracting data from tags of a HTML document [MW15, p. 33]. The general idea of storing all incoming raw data centrally resembles the concept of a data lake (cf. Section 2.2.3). The batch layer is used for, as the name suggests, batch processing. Specifically, Marz and Warren envision that this layer computes batch-based queries on the master dataset and provides these queries to the serving layer for consumption. Computations can be executed on-demand or as precomputation [MW15,

p. 83]. Updating is incremental or a full recalculation as needed [MW15, p. 88f.]. This batch-based processing is explicitly related to MapReduce, but on a higher level of abstraction [MW15, p. 83]. The latter refers to the definition of pipelines to eliminate shortcomings of explicit programming in MapReduce (E. g., as offered by Apache Beam[10], cf. Section 2.2.4). This would allow for linear scaling, whereas non-linear scaling is deemed suboptimal [MW15, p. 92].

Serving layer. Here, data from the batch layer is indexed and provided ("served") through queries from this layer [MW15, p. 177]. One important assumption is that databases in the serving layer are not required to handle small updates (i. e., writes) and can thus be "simple" [MW15, p. 177], which refers to both the primary data model as well as consistency guarantees. Simple data models and relaxed consistency management lead to more resources being available for serving larger portions of data [Mar15, p. 185]. Small updates are delegated to the speed layer of the architecture [Mar15, p. 185]. The recommended technology for this layer should be distributed [MW15, p. 181], especially for the aforementioned index creation. Marz and Warren see *throughput* and *latency* requirements as key determinants to choosing an indexing strategy, where higher cost of building an index must be weighed access higher latency for complex queries on batch layer data [MW15, pp. 181ff.]. The serving layer reads data in bulk (large batches) from the batch layer and builds its indexes afterwards [MW15, p. 185]. For read-access, it is assumed that the serving layer must only randomly read requests accessing smaller portions of data [MW15, p. 185]. Possible technological choices for this layer include distributed NoSQL data stores, whereas ElephantDB[11], a key-value store, is explicitly mentioned (cf. [MW15, pp. 196ff.]).

Speed layer. The final layer is tasked to tackle low-latency updates [Mar15, p. 207]. It is positioned in parallel to the serving and batch layer. Incoming data is pushed to both batch and speed layers, so that the speed layer can

[10] [MW15] mentions another tool for pipeline definition, JCascalog [MW15, p. 115], but Apache Beam was not yet published when the book was published.

[11] Written by Marz, one the book's authors, but the tool has not seen any development activity since 2014 [188].

focus on latency and data can become part of the master dataset. Incoming queries can be answered by the speed layer, if data is not yet available in the serving layer [MW15, p. 208]. Views on "recent" data are computed here, driven by recency and latency requirements (i. e., how fast a view needs to be updated when new data arrives) [MW15, p. 209]. Low-latency updates are delegated to this speed layer and are immediately or eventually merged back to into the master dataset (cf. [MW15, pp. 216f.]). To enable low-latency, Marz and Warren promote incremental algorithms [MW15, p. 207]. The aforementioned separation of batch and speed layers is seen as crucial improvement over traditional RDBMS-based architectures, where the latter would only constitute a speed layer and be hindered by complex operations [Mar15, p. 208]. As storage for this layer NoSQL data stores are promoted, as they would enable fast reads and writes of small amounts of data through their indexing structures, while fulfilling fault-tolerance and scalability requirements of Big Data applications. Apache Cassandra is explicitly mentioned [MW15, p. 211]. As result, data volume on this layer is considerably smaller than on the batch layer [MW15, p. 212]. When updates are asynchronously processed, stream processing should be supported as well, both via micro-batches or per real-time steaming (cf. Section 2.2.6) [MW15, p. 229]. Apache Storm and Kafka are used as an example for an Stream Processing Technology in the speed layer [MW15, p. 242], while Apache Spark (Streaming) is also listed as alternative for micro-batching [MW15, p. 282].

Evaluation

The Lambda Architecture adds both to the notion of Big Data reference architectures as well as architectural patterns in particular. As reference architecture, it depicts roles alongside data storage and data processing activities (and partially, data analytics) of a value chain. However, no particular details are given on data generation and various types as well extended forms of data usage and analytics. The reference architecture is technology-independent, as no technology details are part of the model. However, HDFS, MapReduce, NoSQL data stores, and SPTs (Apache Storm, Spark Streaming)

are promoted as exemplary technologies and tools to implement said architecture. While a batch layer should contain all incoming data for long-term storage, an indexed serving layered is focused on enabling access to users. A speed-layer using streaming or other low latency technology aids this by enabling small updates and low latency access to data. Notably, this reference architecture is not focused on analytics in particular, but on generic Big Data processing.

Apart from the architectural side, Marz and Warren also propose the conceptual foundation to allow both streaming, batch, and read-focused layers to become better to manage by expressing data transformations as function of the original data. Even ETL processes could be expressed as function over raw data.

3.4.6 SOLID Architecture by Martínez-Prieto et al.

Martínez-Prieto et al. propose[12] the SOLID architecture designed as "high-performance architecture for real-time systems consuming RDF data" [CMPF13, p. 46]. Their reference architecture is designed under the assumption that bringing heterogeneous data together is more crucial than volume or velocity alone. Therefore, the authors suggest RDF[13] as graph-based for organizing (semantic) data. RDF triplets consist of subject, predicate, and object and allow linkage between RDF objects. Further, they build on RDF/HDT[14], a binary compression format for RDF maintaining searchability while reducing storage.

Contribution Description

The SOLID architecture adapts properties of aforementioned Lambda Architecture (see Figure 3.15). Its name is derived from the name of its layers (*Service-OnLine-Index-Data* [CMPF13]). While most data is kept in a large

[12] The original proposal is [CMPF13], while [MPCAF15] is a later version with more details and enhancements.

[13] https://www.w3.org/RDF/.

[14] http://www.rdfhdt.org.

data store, a faster form of storage is used, namely a temporary buffer for very recent data. Both forms of storage can be accessed by a front-facing layer, which handles queries. Here, additional layers for indexing data and moving and integrating data into the main data layer are added. The general organization allocates certain tasks mainly within data storage and data processing realms (and partially within data acquisition). Specific technologies are not part of the model itself. The five layers (see Figure 3.16) are as follows[15]:

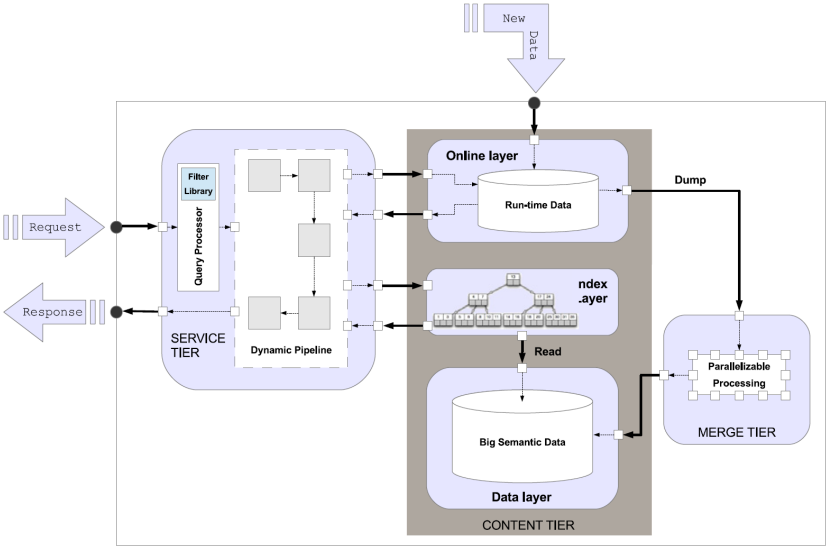

Figure 3.16: SOLID Architecture by Martínez-Prieto et al. Source: [MPCAF15, p. 67].

Online Layer. This initial layer receives incoming data in "real time" and stores it as buffer as long it is not merged into the data layer. Queries for recent data by the service layer are answered by this layer. Therefore, support

[15] Two of them are labeled tiers in the newer publication [MPCAF15].

for indexed structures for querying as well as fast write operations to cope with incoming data simultaneously are required. The authors suggest NoSQL data stores or native RDF stores for this layer. [CMPF13, p. 51]

Data Layer. The data layer is tasked with storing data as binary RDF. Notably, all data in the data layer is seen as immutable [MPCAF15, p. 67], similarly to the Lambda Architecture (see Figure 3.15). New data enters this layer through batch insertions and RDF triplets should be stored using a compressed RDF/HDT format.

Merge Tier. To move and, if needed, transform incoming data into RDF, a merge layer ("tier") is needed. After transformation, data is merged into existing data from the data layer. The authors recognize the computational effort required for these and suggest a MPP capable processing software and specifically name MapReduce and other computations based on it as alternative. Due to the immutability of the data layer, a complete reconstruction in the most optimal data organization format is done. [CMPF13, p. 50] [MPCAF15, pp. 68f.; 78]

Index Layer. The index layer uses indexed data structures that allow to answer queries on large volumes of data in a quick manner [MPCAF15, p. 68]. The authors' intentions are to tackle volume and velocity challenges with such indexed data. Importantly, this particular layer relies on capabilities of data to build such structures while remaining compressed [MPCAF15, p. 68]. When integration into the data layer is completed, the indexes must be updated accordingly. Specifically, the index layer should be tuned towards supporting a dedicated query language for RDF data, SPARQL (cf. [MPCAF15, pp. 70ff.]). To implement these functionalities, data stores offering appropriate index creation are needed. In this case, special triplestores for RDF are suggested, e. g., Apache Jena TDB (cf. [MPCAF15, pp. 75f.]). This is similar to the Lambda Architecture, where NoSQL data stores are recommended for building indexes for serving data efficiently.

Service Tier. Requests and responses are handled by a mediating service layer, which accepts queries for RDFs. SPARQL (cf. [MPCAF15, pp. 70ff.])

is used to query data and to define pipelines for requests and processing RDFs [MPCAF15, p. 70]. Layers behind it, namely data and online layer, are inaccessible to requests from users and machines.

Evaluation

Essentially, the SOLID architecture adds insight about principles of the Lambda Architecture and its usage in a Big Data context with semantic data. Thus, a separate layer for small updates is proposed, while most of data resides in a simple, potentially distributed data store, on top of which an indexed access is built. Queries can access historical data or recent data as needed. However, an empirical evaluation adds additional insights into technological decisions. Another aspect is that, like in the Lambda Architecture, a generic Big Data system is assumed, albeit with semantic data in focus here, and not an analytic system in particular.

For instance, triplestores are suggested for the online layer for incoming data as temporary buffer. Generally, this is similar to suggesting NoSQL data stores for the speed layer of the Lambda Architecture, as the distributed forms of storage allow indexed access to data, while maintaining low latency as long as data size remains within a certain threshold. Martínez-Prieto et al. empirically evaluate both triplestores as well as an in-memory option (also using a triplestore) for the online layer [MPCAF15, p. 77].

3.4.7 Analysis

After the six contributions to Big Data reference architectures have been described and discussed separately, a comparative analysis follows. The goal of this analysis is to assess the scope of the Big Data reference architectures with respect to the unified Big Data value chain (cf. Section 3.2.2). The intention behind this is to reveal the activities a reference architecture covers, i. e., its scope. The more "complete" a Big Data reference architecture is with respect to the value chain, the better it can be used to guide formulation of a unified BI reference architecture, which is a crucial part of this work's contribution. Additionally, the level of detail of the models' components

is evaluated with respect to their core functionality and the data that is used. As the construction of a concrete architecture based on a reference architecture includes an element of refinement, outlining various levels of detail illustrates the variety in levels of abstraction in reference architectures as pointed out in Section 3.1. This can help in selecting the levels of detail necessary for this work's contribution in Chapter 4. Finally, the inclusion of technologies in different levels of abstraction is examined, including to which extent guidance is provided to map these technologies to abstract elements in the respective Big Data reference architecture while constructing a concrete architecture. The results are summarized in Table 3.1. In the following, the attributes, which are examined, are explained and illustrated using the six contributions:

Purpose. As described by Angelov et al., the purpose of a reference architecture can either be *facilitation*, i. e., aid the creation of a customized architecture, or *standardization* [AGG09]. Here, NBDRA [NIS15b] is proposed as means of standardization, while the others are typical reference architectures with template character.

Consideration of Value Chain Activities. As an architecture generates value, which is depicted by means of a value chain. For this attribute, it is examined to which extent a reference architecture is aligned to typical value chain activities (cf. Section 3.2.2).

> **Explicit.** A reference architecture explicitly contains structural elements or relationships, which clarify a connection to value chain activities. This is fulfilled by four of the six models. Their reference architecture models have explicit labels which denote activities like "data processing" or "data analysis".
>
> **Implicit.** The reference architectures [MPCAF15] and [MW15] do not explicitly denote a relationship to value chain activities inside their models. Nevertheless, this connection can be derived implicitly, e. g., by considering the data flow.

Value Chain Activities. For each activity of the unified Big Data value chain in Section 3.2.2, it is checked if it is depicted in the respective reference architecture models. For example, if a reference architecture depicts an abstract storage facility, the attribute *Data Storage* is marked with ✔. If an activity is only implicitly covered, it is denoted with (✔). In this case, [MPCAF15] and [MW15] depict only an abstract form of data generation, i. e., that "some" form of new data enters the system. Moreover, both aforementioned models only consider an abstract form of data processing and data usage by depicting generic "queries" by the user. This also means that *Data Analysis* is not part of these reference architectures. Furthermore, the Lambda architecture only implicitly shows any form of *Data Acquisition* [MW15]. Notably, the combined value chain activity of *Data Acquisition and Processing* is separated as technologies for these activities might differ.

Architectural Patterns. This attribute denotes which kind of architectural patterns (cf. Section 3.1) are ingrained into the model. This influences the variety of patterns, which are "passed on" during the construction of concrete architectures.

> **Static.** There is one fixed or a fixed selection (i. e., static) of architectural patterns in the reference architecture. For instance, the Lambda architecture [MW15] only considers two pathways for data: its batch layer with storage and processing functionality and its speed layer, which can quickly disseminate data. As [MPCAF15] is related to the Lambda architecture, it inherits these architectural patterns.
>
> **Dynamic.** Dynamic architectural patterns mean that the reference architecture intends to allow any form architectural patterns in the specific architectures created using it. For instance, the reference architecture by Pääkkonen and Pakkala permits flexible data flows between each of its layers [PP15, p. 5]. The reference architecture by Oracle contains explicitly back- and forward flows between its three main areas of the architectures, which allow for flexible architectural

patterns. The reference architecture by Geerdink sees every element in it as "optional", enabling dynamic patterns [Gee13, p. 74].

Functionality Detail. Functionality in a reference architecture delivers value for data usage. Typically, these are analytics or processing functionalities. Depending on the reference architecture, these can be depicted in component form at different levels of detail. The assessment of detail level is derived from Angelov et al. [AGG09, p. 3] and Greefhorst et al. [GKV06, p. 110], who assess component and element detail in reference architectures and frameworks. Here, the level of detail is assessed as *None*, *Low*, *Medium*, or *High*.

None. When functionality is completely generalized without any details, no further distinction of its contents or roles is possible. Here, the Lambda and SOLID architectures do not differentiate functionalities and depict them only as arbitrary *query* that enters the system and is answered [MW15, MPCAF15].

Low. In this abstract form, functionality is depicted according to value chain activities without many details. For instance, the reference architecture by Geerdink only considers abstract value chain activities as functionality such as "Ingest" or "Analyze" [Gee13]. Similarly, the NBDRA does not depict further details of functionality, which is unsurprising as its purpose is standardization [NIS15b].

Medium. A medium level of details adds to the generic value chain activity. Here, a high-level functionality class is used to denote a content-based categorization of respective functionalities, such as "Advanced Analytics", "Text and Spatial Analytics" [HPS+16], "Stream processing", or "Transformation" [PP15]. These classes add more detail to the conducted functionality and allow for disambiguating them.

High. A high level of detail would point to more specific form of functionality, such as "classification", "dimension reduction", or "prediction" (e. g., cf. Section 2.3.3). None of the examined models depicted functionality with this level of detail.

Data Characterization. Naturally, a reference architecture refers to abstract forms of data, which is ingested into an architecture. Here, the specific characterization of *data* outlines by which dimension this abstraction is conducted. Characterization is not assessed using a particular scale, but qualitatively.

> **None.** An arbitrary, generalized data source is not further disambiguated and simply denotes that unspecific "data" enters an architecture.
>
> **Origin.** Here, data is characterized by its origin, e. g., as done by [Gee13], who disambiguates *enterprise* and *open* data.
>
> **Variety, Velocity.** Oracle [HPS+16] and Pääkknonen and Pakkala [PP15] characterize data by variety and velocity.

Technology Detail. Similar to the assessment of functionality detail, the technology detail aims to point how abstract or detailed technology is included in the respective reference architecture models[16]. It should be noted that the depicted Big Data reference architectures attempt to be technology-independent in general. Nevertheless, [Gee13], [HPS+16], and [PP15] explicate technologies in different levels of detail in form of models. [HPS+16] and [PP15] contribute separate models which depict technology, which can help to build a customized architecture during the main Big Data reference architecture. For this attribute, the level of detail is assessed based on based on [AGG09, p. 3] and [GKV06, p. 110] as *None*, *Low*, *Medium*, or *High*.

> **None.** Specific details of technology are not discernible. For instance, Marz and Warren do not depict any specific technology in model-based form. The Lambda architecture only features an abstract notion of, e. g., storage. Selected recommendations for technology for custom implementation are only given in textual form.

[16] Simple examples or selected textual recommendations are not considered. The assessment is only done using visual models.

Low. A low level of detail means that technologies are depicted as abstract technology class based on technical properties, which contain several more specific technology classes such as RDBMSs. The NBDRA features "Indexed Storage" and "File Systems" for data organization purposes. "Indexed Storage" can refer to both NoSQL data stores as well as to RDBMSs.

Medium. Grouping technology into distinct named classes represents a *medium* level of detail. For instance, Oracle [HPS+16] depicts "RDBMS" or "NoSQL". Here, the classes of RDBMSs and NoSQL data stores can be distinguished by several dimensions.

High. Mentioning instances of specific technologies (*tools*) such as MySQL, PostgreSQL is considered a *high* level of detail. These instances can be applications, appliances, or other named software systems. Pääkkönen and Pakkala [PP15] offer a technology map, which organizes tools into technology classes.

Technology Mapping. This final attribute expresses whether guidance is provided to select or to map given technologies to the proposed reference architecture to customize it. A technology mapping is fully provided ✔, if a structure of suitable technologies can be explicitly associated with the elements of a reference architecture by means of a model. As the Lambda and SOLID architectures and the Big Data reference architecture by Geerdink only name selected examples for technology mapping, this attribute is assessed as ✖. The NBDRA does not mention specific technology at all and is also marked with ✖. The other reference architectures provide more comprehensive lists or even models with technology of various detail, but neither of them offers an explicated mapping of these onto the elements of the respective reference architecture, although selected details and examples are mentioned. Thus, these are denoted with only partial fulfillment of this task (✔). For instance, Oracle gives generic advice on when to employ a stream processing solution and illustrates Big Data use with three use cases.

Table 3.1: Big Data reference architecture analysis summary. Sources: see table header.

Reference Architecture	[MPCAF15]	[Gee13]	[MW15]	[HPS+16]	[PP15]	[NIS15b]
Purpose	Facilitation	Facilitation	Facilitation	Facilitation	Facilitation	Standardization
Consideration of Value Chain Activities	Implicit	Explicit	Implicit	Explicit	Explicit	Explicit
Data Generation	(✔)	✔	(✔)	✔	✔	✔
Data Acquisition	✔	✔	(✔)	✔	✔	✔
Data Storage	✔	✔	✔	✔	✔	✔
Data Processing	✔	✔	✔	✔	✔	✔
Data Analysis	✖	✔	✖	✔	✔	✔
Data Usage	(✔)	✔	(✔)	✔	✔	✔
Architectural Patterns	Static[viii]	Dynamic	Static[ix]	Dynamic	Dynamic	Dynamic
Functionality Detail	None	Low	None	Medium	Medium	Medium
Data Characterization Dimension(s)	None	Origin[x]	None	Variety, Velocity	Variety, Velocity	None
Technology Detail	None	Medium[xi]	None	Medium	High	Low
Technology Mapping	✖[xii]	✖[xiii]	✖[xiv]	(✔)[xv]	(✔)[xvi]	✖

[viii] Batch/Speed layer. [ix] Batch/Speed layer. [x] Enterprise, Open Data. [xi] Mixed with selected specific technologies, which are resemble a *high* level of detail. [xii] Focused on technology evaluation related to semantic data; Only textual advice is given. [xiii] Generic examples of Big Data technologies are provided. [xiv] Only textual examples are given. [xv] Technology classes, vendor-specific technology list, and sample use cases provided. [xvi] Technology class and and technology lists with use cases provided.

Implications

In summary, several findings can be pointed out. First, the NIST Big Data Reference Architecture from the NIST [NIS15b] features a "total" view of the spectrum of solutions in the Big Data area. While it is not meant for facilitation, it helps to position a reference architecture within this solution spectrum and provides several hints for technology usage. While the Lambda [MW15] and the SOLID [MPCAF15] architectures exhibit template character, they are focused on a particular set of patterns, which divides data flows into a "batch" and a "speed" layer. In particular the Lambda architecture advocates a specific principle in working with data, which inhibits its universality for a unified BI reference architecture. In spite of this, a selection of technologies is recommended based on use case criteria, which might be useful for general technology selection. While the Big Data solution reference architecture by Geerdink [Gee13] is designed with a dedicated modeling language, which the other contributions do not use, it depicts functionality with very few details an in a highly abstract manner. While it is focused on a certain set of data sources, it also allows for dynamic patterns, as its elements are optional. This further underlines that a universal architecture and its development process should be more flexible and allow for dynamic patterns, which are aligned to the unified Big Data value chain. However, such flexible patterns might necessitate more details at the functionality level to select technologies for an architecture. Finally, the contribution by Oracle [HPS+16] features more flexible patterns and offers a separate "capability map". However, this map is not directly connected to the reference architecture and mixes several types of elements, such as technology classes, specific technologies, functions, and approaches. This is another indication that a structured mapping process as part of the construction of a customized BI architecture is required.

3.5 Big Data Architectures in Practice

Analyzing the use and inception of specific Big Data architectures can reveal further insights into the reasoning behind building them.

Specific requirements are the root for the construction of an architecture. Based on these requirements technologies with specific properties are chosen. Describing specific implementation sheds further light on this selection process and might outline additional use cases for various technologies.

The goal of this section is to enrich generic properties and use cases of technologies outlined in Chapter 2 as well as during the analysis of Big Data reference architectures in the previous section. Besides that, architectural patterns employed in these solutions are outlined using the unified Big Data value chain proposed in Section 3.2.2. As far as documented, each value chain activity of each identified architecture is described. Extracted reasons for technological choices are recalled in the subsequent section Section 3.6 as necessary.

Here, the specific Big Data architecture implementations by **Netflix** [Ama13], **LinkedIn** [SKG+12], and **Twitter** [MDL+13] are examined. They represent a selection from the result of the literature search and are used to illustrate the topic. These companies introduce novel solutions to the scientific community and contributed several tools, which are now open source and widely adopted (e. g., Apache Storm by Twitter [TDB+14]). Therefore, these cases are suitable to add details to technological choices from the Big Data solution spectrum.

Netflix [Ama13]

Amatriain describes the architecture at Netflix for film and TV series recommendations.

Data Velocity. Both "complex" offline batch and "real-time" streaming data are used as input. [Ama13, p. 1]

Data Volume. Netflix has "billions of ratings", increasing by millions each day. Moreover, there are millions of personal queue items added and items

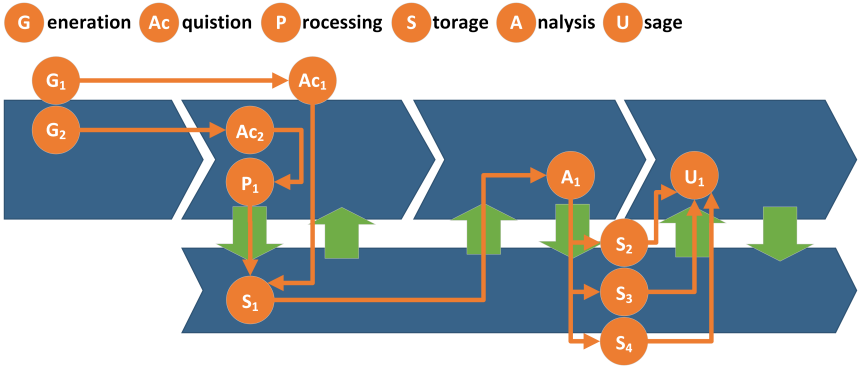

Figure 3.17: Architectural Pattern used at Netflix [Ama13], which indicates information flows as usage of an architecture between components allocated to activities of a value chain.

search per day [Ama13, p. 3]. Moreover, similarly sized play, browse, and search histories can be evaluated [Ama13, p. 1].

Data Variety. Data items at Netflix have "rich meta-data". There are ratings, members, movies, impression data (GUI interaction with recommendations) [Ama13, p. 3]. There is also social network data, e. g., connections on Facebook. This is complemented by external data such as box office performances or movie reviews. Other personal metadata includes data items such as location, language, or demographics [Ama13, p. 4].

Goals. One goal of the system is to deal with large streams of data to extract information for personalizing their service through a recommender system [Ama13, p. 1]. Their system should especially provide explanations to why a recommendation, e. g., for a specific movie, was given [Ama13, p. 2]. Additionally, it should provide a ranking of recommendations, including sorting, in real-time (i. e., as soon as a user invokes a movie landing page) [Ama13, p. 2]. Their software architecture should be "scalable" and "reactive" [Ama13, p. 4]. According to them, a key challenge is to intelligently combine offline and online computation models with machine learning.

Solution and Rationale. "Nearline" computation is used between between offline and online computations. Model training can use offline, nearline, or real-time (online) computation. For real-time, latency is the key factor, which might require additional infrastructure. The architectural usage pattern of the solution is depicted in Figure 3.17.

> **Data Acquisition.** Netflix uses a custom publish/subscribe data flow system, however, it is noted that Apache Kafka covers most its use cases as well. The reasoning is that subscribers should be notified, when long running queries are ready [3]. This tool is used for both batch and real-time use cases. For initial data acquisition, a Hadoop-based log collection tool, Apache Chukwa[17], is used. This tool has similar goals as Apache Flume, but a slightly different implementation approach. For instance, Chukwa targets latencies for near real-time in the range of minutes, whereas Flume operates in second spans (cf. [RK10]). [3]
>
> **Data Processing and Analysis.** Batch processing is used for most model computation to save time when delivering results. Querying on processed data is done via Apache Hive and Pig, which should support several storage repositories. To handle incoming interaction events as streaming data from various Netflix apps (e. g., smart TVs or phones), a custom stream processing system is used, which is supposed to be similar to Apache Storm. Netflix notes that their custom implementation was chosen due to internal requirements, which are not disclosed. As most of the recommendations are precomputed, only limited enriching is done to recommendations through incoming real-time signals in order to serve requests to users in real-time. Implicitly, it can be assumed that soft real-time requirements are in place for this layer, as recommendations need only be served as answer to a user's page request via Internet. [3]
>
> **Data Storage.** Distributed storage is employed due to large data size. They use HDFS as well as cloud storage (Amazon S3) and Apache Cas-

[17] http://chukwa.apache.org.

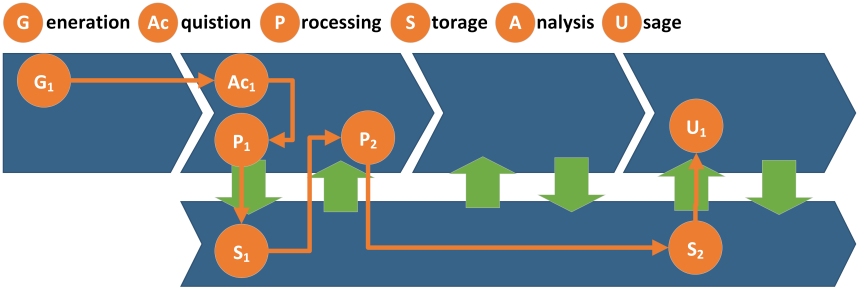

Figure 3.18: Architectural Pattern used at LinkedIn [SKG+12], which indicates information flows as usage of an architecture between components allocated to activities of a value chain.

sandra. RDBMS MySQL is used to store structured information and because of its "general-purpose" querying capabilities. However, its scalability is deemed too limited due to this. Thus, Apache Cassandra is used as distributed key-value store. For Netflix, Cassandra does not perform well in constant, write-intensive scenarios, so they use EVCache[18] as alternative in-memory key-value store. [3]

LinkedIn [SKS13]

Sumbaly et al. describe the architecture of the data pipeline for data mining applications at LinkedIn[19]. LinkedIn offers an online job portal and was recently purchased by Microsoft [268].

Data Variety. LinkedIn has constantly evolving data models, because functionalities are constantly added to their website. Moreover, incoming data from there is streaming event data (e. g., a user opening a page) as well as structured master data (e. g., company, member, and other profile data) from inside an Oracle database [SKS13, p. 1127]. Generated recommendations are stored as key-value based data (e. g., recommendations for a member are

[18] https://github.com/Netflix/EVCache.
[19] https://www.linkedin.com.

placed in a value according to his member key), because data models should remain flexible [SKS13, pp. 1129].

Data Velocity. Streaming data from events is high-velocity data with up to 200,000 messages per second [SKS13, p. 1128]. On the other hand, there is resting, structured data in the Oracle database. Analytics need to be delivered in near real-time, i. e., as soon as a user has loaded a page.

Data Volume. It is noted that the accumulated streaming data amounts to more than 100 TB and is "several orders of magnitude" larger than "data in databases" [SKS13, p. 1127]. In a paper describing their OLAP system *Avatara* [WSR+12], the authors state that each data cube for each of LinkedIn's 160 million members[20] is between few and up to several tens of MBs large, which puts this data volume at least in TBs range. Conclusively, Sumbaly et al. argue that horizontal scaling through distributed systems is necessary.

Goal. LinkedIn aims to develop an architecture to "extract insights and build product features from massive amounts of data". The system should enable several use cases at LinkedIn such as recommendations (e. g., collaborative filtering), "news feed updates, email digesting, and descriptive analytical dashboards" [SKS13, p. 1125].

Solution and Rationale. LinkedIn distinguishes between ingesting data and analyzing it and making the analysis results available to their site in various formats. The architectural usage pattern of the solution is depicted in Figure 3.18.

> **Data Acquisition.** Streaming data is acquired through Apache Kafka (cf. Section 2.2.6), which LinkedIn initially developed for their specific usage (high-velocity data, which builds up to large volumes and needs to be routed in a distributed manner). Kafka provides this data to MapReduce jobs for ETL ingestion, which run every 10 minutes and load data Kafka has stored/cached. During acquisition, data is checked against flexibly defined schema constraints. [SKG+12, pp. 1127f.]

[20] As of 2012.

Data Storage. This ingestion loads data into HDFS storage. Each day, an aggregation and garbage collector aggregates processed data and removes unneeded older data to free storage [SKS13, p. 1128]. For output recommendations, Voldemort (cf. Section 2.2.2) is used as key-value store. To provide descriptive dashboards, OLAP cubes are precomputed with MapReduce to enable high data throughput. These cubes are served with low latency via Voldemort to deliver them directly at page view time. Avatara allows to access Voldemort with SQL-like queries [WSR+12].

Data Processing. As "primary processing interfaces", Hadoop MapReduce and Apache Hive are used together with Apache Pig [SKS13, p. 1128]. Using Apache Hive, ingested streaming data is often accessed by time with SQL-like queries [SKS13, p. 1129]. Moreover, more complex workflows are built using Pig Latin as query language instead of native MapReduce. LinkedIn uses a self-developed workflow management tool throughout their whole cluster, which is similar to Apache Oozie (cf. Section 2.2.5) [SKS13, p. 1129]. Exploration and production environment, using HDFS and MapReduce, are separated. To push stream data back to the front-end, Apache Kafka is employed. This also the case for results produced offline [SKS13, p. 1132].

Data Analysis and Usage. Mostly, specifics of data analytics methods or tools are not detailed, besides generic use cases such as predictive recommendations, collaborative filtering and association rule mining, text analytics (e. g., synonym detection) OLAP cubes (cf. [WSR+12]) for standardized reports (i. e., data description methods, cf. Section 2.3.3). It is implied that custom data pipeline workflows are utilized for analytics as well [SKS13, pp. 1131ff.].

Twitter [MDL+13]

[MDL+13] describes a real-time architecture of microblogging service Twitter for "related query suggestions" [MDL+13, p. 1148] and spelling correction (termed "search assistance"). Suggestions in form of related search terms

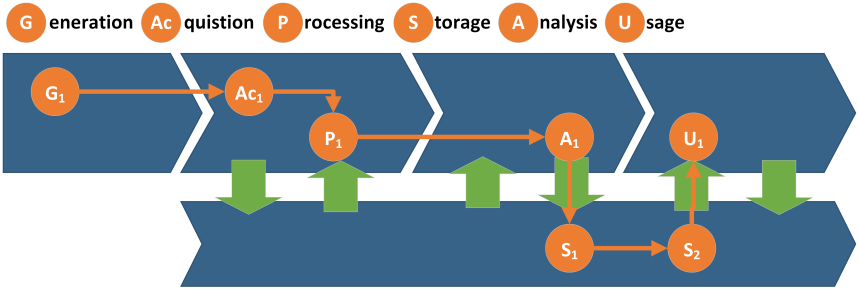

Figure 3.19: Architectural Pattern used at Twitter [MDL+13], which indicates information flows as usage of an architecture between components allocated to activities of a value chain.

are shown as a popup list while users type their desired search term into a search field.

Data Velocity. Data at Twitter is produced at high velocity by its numerous users (e. g., as pointed out in Section 2.2.1). Twitter's requirement is that no later than 10 minutes after a new trend emerges, it should be available as search query suggestion (i. e., combined data and analysis latency is less than 10 minutes). Moreover, incoming search requests need to be answered it real time.

Data Volume. Each day, incoming events (e. g., client interactions through various apps) amount to volumes in a hundred TB range of unprocessed, aggregate data [LLL+12].

Goal. In general, the desired search assistance system should functionally be able to "execute complex queries" and user-defined functions with "with varying latency requirements". Computations should work on large temporal time windows (cf. sliding windows in Section 2.2.6) as well small time-windows for real-time analyses [MDL+13, p. 1154].

Solution and Rationale. Generally, data is acquired and aggregated with a real-time log processing system from Twitter's "firehose"[21], a streaming API acquiring and making available all tweets from the system. Calculated statistics are stored in-memory before ranking algorithms calculate results for related query suggestion (e. g., rank of suggestions). Data is then stored in HDFS and served with a distributed real-time capable system, which pulls data as needed from HDFS. The architectural usage pattern of the solution is depicted in Figure 3.19.

Data Acquisition. Twitter uses an in-memory acquisition tool to query streaming data from Firehose (for Tweets) and streaming input for index-search query data (cf. [BGL+12]). The latter is created by a service which ingests search queries and using an unnamed real-time ingestion system. It ultimately uses a distributed processing system to create searchable and queryable in-memory index structures in a separate "Blender" system, which is not further detailed [BGL+12].

Data Processing and Analysis. Exploratory analyses are conducted using MapReduce with Apache Pig. Data on HDFS can be combined with incoming streaming data for this. Analysis for search assistances focuses on methods for text analytics (cf. Section 2.3.4). In this case, co-occurrences between mentioned terms in tweets need to be analyzed to offer ranked suggestions. The more a term is mentioned, the more are weights in their models adjusted [MDL+13, p. 1153].

Data Storage. Persistence of both analyzed results and incoming data (after aggregation) is persisted on HDFS due to scalability and fault-tolerance. Further, low latency is not needed for it. Aggregated data and results are persisted every five minutes. This is also used to create few, larger files for HDFS [MDL+13, p. 1151]. Analysis-relevant statistics created by aggregations as well as other measures are stored in three undisclosed in-memory stores. It is admitted that limited main memory size leads to trade-offs in statistical models [MDL+13, p. 1154].

[21] Nowadays, it is available for selected enterprise customers as PowerTrack API [387].

> An undisclosed, but scalable in-memory based cache is used to provide results permanently persisted on HDFS to search query issued on Twitter front-ends. Once per minute, it pulls necessary data from designed HDFS locations [MDL+13, p. 1153].

3.6 Selection Criteria for Analytical Architectures

To address the research questions, in particular RQ_2, it is necessary to know under which contingencies, which technology is used in a BI architecture. In this section, selection criteria for technologies in an architecture are summarized. These are used later as input for the construction of the proposed research artifact in Chapter 4. *Technological* criteria directly address the technical properties of technologies such as throughput and latency. Based on properties and characteristics of technologies outlined in Chapter 2 and the analysis of DWHs advancements in general, selected literature on Big Data reference architectures, enriched by practical use cases, a set of technological selection criteria is synthesized in the following. The analysis revealed primarily technological criteria. Criteria with respect to *organizational*, *economical*, and *regulatory* perspectives are thus elaborated on in the context of the contribution (cf. Section 4.4)

For several technologies, or classes of technologies, criteria are identified, which guide their selection for a BI architecture. These criteria address one or more challenges of Big Data, related to its 5V properties (cf. Section 2.2.1). Notably, most challenges are related to the three technical properties volume, velocity, and variety. Apart from that, the use of specific technologies is associated to certain value chain activities.

In total, the findings regarding the technical properties of technologies as outlined in Chapter 2 are consistent to their usage and selection process examined in this chapter. For example, the pictured advantages of HDFS such as its ability to scale horizontally is mirrored by such usage in both Big Data reference architectures and architecture implementations.

Storage & Processing: Indexed Data Access

Indexed data access is provided by storage systems such as RDBMSs and NoSQL data stores. These systems create an index over data, which allows to explore data structure and linkages using queries. In contrast to indexed data access, file systems store data as-is and can only retrieve it that way. Although enabled by a data *storage* function, the core of indexed access resembles a *processing* functionality with respect to a value chain.

A main decision criterion for indexed data access it the desire of *Querying Data Structures and Linkages* [NIS15b, pp. 23f.] [MW15, p. 177] [MPCAF15, p. 68], which is given by the nature of indexed data access. Although differences with respect to the complexity of data structures exist, choosing the proper storage format for the data at hand is enabled by the variety of storage options offered by RDBMSs and NoSQL data stores. These help to tackle the challenge of data *variety* as many forms of semi-structured data and structured data as well are addressed. As noted by the NIST in [NIS15b, pp. 23f.], it is not just the structure and linkages inside the data which are relevant, but also the actual access interfaces offered by a respective application. For instance, the query language SQL is widely used due to its ability to express complex and ad-hoc queries with various functionality (cf. Section 2.1.1). For instance, Netflix also employs MySQL due to its "general purpose" querying capabilities [3]. As stated in Section 2.2.2, NoSQL data stores in general possess simpler access interfaces, which do not offer the complex capabilities of SQL, but only, e. g., a limited set of "SQL-like" functionality or plain programmatic APIs (cf. Section C.1.2).

Moreover, index data access is chosen to reduce the *Analysis Latency*, i. e., the response time by which data from an already created structured indexed can be retrieved as specified in a query. Both Mishne and the NIST note that especially in-memory technology (cf. Section 2.2.7) combined with indexed data storage contributes to tackling the challenge of data *velocity* by enabling retrieval of data in real-time respectively near real-time [NIS15b, p. 22] [MDL+13, pp. 1153f.]. Marz and Warren add that simpler data models in general (e. g., key-value based) help to decrease response time [MW15,

p. 185]. Notably, Oracle recommends NoSQL data stores in general for "real-time" retrieval [HPS+16, p. 14].

Storage: Distributed Storage

The aspect of distributed storage only refers to criteria for the *storage* activity of a value chain.

Distributed *indexed storage* refers to NoSQL data stores and RDBMSs which are horizontally scalable (cf. Section 2.2.1). They fulfill requirements on *scalability*, which tackles the challenge of a high data *volume*. This is confirmed by several sources such as [MW15, p. 83] and by the inherent properties of the technologies in general (cf. Section 2.2.2 and Section 2.2.7). In particular, such solutions allow for high performance by offering a large *Throughput* (cf. [MW15, p. 181ff.]), which constitutes another decision criterion for storage systems. Apart from that, another criterion is the focus with respect to the CAP properties (e. g., AP or CA), which needs to be tuned according to specific application requirements (cf. Section 2.2.2, [MW15, p. 211]).

Similarly, the argument of scalability applies to distributed *file systems* such as HDFS (cf. [MDL+13, p. 1151] [NIS15b]; cf. Section 2.2.3). For example, Netflix uses HDFS for distributed storage of large volumes as well and exploits its scalability [3]. However, such file systems do only offer *file-based access* and do not possess indexed access or other processing functionalities as RDBMSs and NoSQL data stores do. Any type of format or, in particular, data without any structure can stored "as-is" in file systems. This can meet particular *variety* challenges.

Notably, for both file systems and indexed storage, a trade-off between *performance* (throughput) and *latency* might have to be chosen (cf. [NIS15b]).

Processing: Distributed Processing

Technologies for *distributed processing* leverage distributed workers in a shared-nothing architecture for MPP. This refers to generalized processing on data — in contrast to queries on (pre-)indexed data as outlined before.

For batch-based processing to tackle large *volumes* with no particularity latency requirements, batch processing frameworks such as MapReduce are suitable [HPS+16, p. 15] [MDL+13, pp. 1151f.]. This is consistent with the properties of MapReduce working on distributed storage as detailed in Section 2.2.4. As with file systems, distributed processing with MapReduce on HDFS allows for high throughput of data processing, contributing the said data volume challenge [SKG+12, p. 1130].

When *low latencies* are needed, distributed processing can be tuned towards by using technology working in-memory, such as Apache Spark [HPS+16, p. 17] [SKG+12, p. 1128]. Streaming Processing Technologies in general are well-suited for scenarios involves real-time or near real-time responses (cf. Section 2.2.6).

Acquisition: Batch-based Ingestion and ETL

While the Big Data reference architectures and the architecture implementations discussed do not employ traditional ETL or DWH technology, the need for ETL-related tasks and other batch-based data ingestion can still be identified. Such data *acquisition* technology focuses on the acquisition of various sources and ingests it. Tools such as Apache Chuckwa (cf. [SKG+12]) and Sqoop work in a distributed fashion and fulfill such extraction and loading tasks. Notably, in data lake approaches, e. g., in [PP15], data is loaded first and the pre-processed (cf. ELT).

Acquisition to Analytics: Streaming Processing Technologies

The handling of *streaming data* implies a different approach than handling resting data due to high *velocity* and real-time or near real-time *analysis latency* requirements. Due to this, a stream processing pipeline can cover the complete value chain from acquisition, to processing, and analytics (cf. [NIS15b, p. 25], [BGL+12]). For instance, in the synthesized reference architecture by Pääkkönen and Pakkala, streaming data incurs separate choices for the same tasks as resting data [PP15, pp. 3ff.]. Moreover, streaming data collection tools such as Kafka are used to horizontally scale the acquisition of

streaming data (cf. Section 2.2.6; [WSR+12], [SKG+12, p. 1128]). Nevertheless, stream processing is not completely separate from other technologies in an architecture, as it might need to add master data for context [HPS+16, p. 17].

3.7 Solution Artifact Requirements

The analysis of analytical architectures clarifies several aspects. As postulated in the beginning, there is indeed no generally agreed upon universal BI architecture yet. At least the documented cases of Big Data reference architectures and concrete architectures focus on technologies and usage patterns, which are novel. Nevertheless, certain facets of traditional architectures, such as ETL or SQL can be identified. This stresses that a contribution of a unified BI architecture is indeed purposeful.

However, the analysis of reference architectures also highlighted the need for more guidance to navigate through the solution space offered by such reference architectures. Although several technologies are suggested based on various criteria — as summarized in Section 3.6 — there was no explicit connection to a reference architecture. Technological choices were mostly described in text, e. g., using example uses.

Hence, this section aims to define specific solution artifact requirements — the *solution objectives* according to the DSRM process — which the contribution of this work should fulfill.

In essence, a unified BI architecture should contain traditional and novel technologies as outlined in this chapter. The reference architecture model must be designed to enable dynamic architectural patterns to be truly universal. Moreover, the reference architecture must be constructed in a way that software selection can be explicated with respect to the model. For the selection of software, the outlined criteria must be taken into account and used to link a given use case to specific technologies. These two artifacts combined answer the respective research questions R_1 and R_2 and form a *method* as solution artifact, which is detailed in the following chapter.

Next, Section 3.7.1 lists specific principles, which refer to further contents of the solution artifacts and to its underlying design using a visual modeling language. Afterwards, Section 3.7.2 specifies the actual list of solution artifacts requirements.

3.7.1 Solution and Design Principles

To ensure high quality of a reference architecture, architecture, process, and other conceptual models, several quality frameworks and guidelines are available (e. g., the SEQUAL framework [LSS94, KLS95] or the 7PMG framework [MRvdA10]). One of those are the so-called *Guidelines of Modeling (GoM)* [Ros03, pp. 58ff][22]. The GoM define six principles to ensure qualitative models:

1. **Correctness[23].** A model should be both semantically and syntactically correct, which means that real-world exemplar is appropriate represented in its structure behavior, while adhering to the syntactic rules of the underlying modeling language. [Ros03, p. 59]

2. **Relevance.** High-quality models depict the relevant parts of their original with respect to the intended purpose. However, this also means that no superfluous information is contained in a model, as this would be irrelevant. [Ros03, p. 59]

3. **Economic efficiency.** To ensure that the cost of following the guidelines strictly does not outweigh its benefits, trade-offs can be made to ensure cost efficiency of modeling endeavors. Rosemann notes that is would be "extremely difficult" in practice. [Ros03, p. 59f.]

4. **Clarity.** In the context of its usage and purpose, a model should be intuitively readable by its users to be actually useful. This necessitates that users can comprehend a given model. [Ros03, p. 60]

[22] Originally proposed in German language as "Grundsätze ordnungsmäßiger Modellierung" in [BRS95].

[23] Name of this and the following principles are abridged. Originally formulated as, e. g., "Guideline of correctness" [Ros03, pp. 59f.].

5. **Comparability.** Models should follow consistent modeling conventions, e. g., with respect to element naming. Consistency should be ensured within a model and between several models.

6. **Systematic design.** As models are a purposeful abstraction of reality, they do not represent all perspectives and facets of the real-world object they refer to. Thus, models should consider other models as part of their system and specify interfaces and relationships to respectively with them. [Ros03, p. 60]

Such principles serve several purposes. They help to manage the variety in the corpus of models by attempting to keep them consistent, which allows to compare models and allow for clear analyses. (cf. [Ros03, p. 60f.])

Besides general design principles for conceptual models, there are also more solution-specific guidelines. Chen et al. propose seven principles, which *Big Data systems* should adhere to. Otherwise, proper "Big Data analytics in a highly distributed system" would not be achievable [CML14]. They can be summarized as following:

1. **Appropriate architectures and frameworks.** Chen et al. underline the importance of a "good and proper" architecture behind the Big Data system [CML14, p. 331]. According to them, a higher-level architecture is required for Big Data systems, as there are many technologies that solve different problems. For data-intensive applications they see it as necessary to "design different and appropriate architectures in the beginning of everything" [CML14, p. 331].

2. **Support for a variety of analytical methods.** Due to the large corpus of methods available of Big Data analytics (cf. Section 2.3), several methods such as machine learning or distributed processing should be made available. This is needed as Big Data tasks would be "complex". This complexity necessitates a variety of methods to be tackled. [CML14, p. 331]

3. **Solutions must be custom fit to the requirements**. Big Data opportunities opened up a wide range of analytical possibilities, which

reaches beyond what a traditional DWH reference architecture encompassed. Chen et al. acknowledge that Big Data tools are not able to cover all these use case simultaneously and may exhibit trade-offs between various properties such as throughput vs. latency (cf. [CML14, pp. 331f.]). This implies that indeed the best tool or combinations of them for a given case need to be found.

4. **Analyses should be brought to the data.** When large data volumes are stored in distributed systems, pulling all data into a separate system for analysis is not feasible. Thus, a "code-to-data" approach should be utilized, where algorithmic calculations are executed where the actual data is located (e. g., as done with MapReduce on HDFS, cf. Section 2.2.4). [CML14, p. 332]

5. **Processing should be distributable for in-memory computation**. Chen et al. assert that in-memory analytics gains popularity as they increase speed vastly and enable real-time analytics. Alluding to the fact that memory capacity on a single system is limited in comparison to secondary storage (cf. Section 2.2.7), they point out that it is necessary to conduct distributed in-memory computations to tackle volume and velocity related challenges at once. [CML14, p. 332]

6. **Data storage should be distributable for in-memory storage.** Combining the two previous aspects, it becomes clear that for in-memory analytics conducted on data residing in-memory to work, data storage must be distributed as well. Chen et al. see additional benefits in leveraging cloud services for that purpose. [CML14, p. 2014]

7. **Coordination is needed between processing and data units**. Shared-nothing architectures promote horizontal scaling capabilities. Nevertheless, coordination of large clusters and providing resource and configuration management, becomes complex if it has to be conducted manually. Thus, Chen et al. see the use of coordination software such as YARN or Apache ZooKeeper (cf. Section 2.2.5) as necessary.

3.7.2 Requirements Overview

Based on the findings in this chapter, several *solution objectives* can be identified, which should be fulfilled by this work's contribution. The objectives are formulated as requirements with respect to the solution (i. e., contribution) and its artifacts. These requirements further consider key aspects of the architecture design principles outlined in Section 3.7.1.

Naturally, the solution objectives are aligned with the research questions stated in Section 1.1. The objectives are more specific with respect to the to-be-designed artifacts and state *how* the research goals are met by the contribution.

SR_1 **The solution shall consist of a BI reference architecture and a development process.**

Mandated by the research questions in Section 1.2, a BI reference architecture as well as an appropriate process supporting the selection of technologies for attaining a customized BI architecture is needed. This process is termed *development process*. While picturing the building blocks of a BI solution in a BI reference architecture is purposeful, there are many more elements to it as to the traditional DWH reference architecture — in particular Big Data and Streaming Processing Technologies. Simultaneously, architectural patterns became more diverse, allowing for different combinations of traditional and novel technologies. To offer guidance, a process to develop a customized BI architecture based on such complex reference architectures is needed.

SR_2 **The BI reference architecture shall display a current selection of novel and traditional technologies and approaches.**

The technologies that are the basis for a customized BI architecture should represent both traditional and novel technologies as outlined in this work such as RDBMSs, ETL, as well as Big Data and Streaming Processing Technologies. Moreover, these technologies should cover typical activities in a unified Big Data value chain, such as storage, processing, or analytics. Chen et al. underline the importance of

offering various analytical methods as well as generic analysis approaches (e. g., distributed processing by being able to bring code to the data at hand [CML14, p. 332]).

SR_3 **The development process shall select technologies based on specific goals and requirements.**

As mandated by RQ_2, guidance offered by the development process should be based on a given use case. The properties of such use case are given by its goals and requirements with respect to a BI solution. These lead to technological selection criteria, which drive the development process. This ensures that the purpose of BI is met, as BI should be aligned to an organization's goals (cf. Section 2.1). *Goal-orientation* further ensures that the typical aims of requirements engineering are met (cf. Section 2.5). Pohl explicates that architectural design requires proper functional and quality requirements [Poh10, p. 314], which is also affirmed by the respective architecture design principles outlined before.

SR_4 **The development process shall explicate the connection between goals and requirements and technological selection by directly relating to a BI reference architecture.**

The selection of technologies based on specific goals and required as stated by the previous requirement SR_3 should be conducted by forming an explicit connection between these while referring to a BI reference architecture. The analysis of the reference architectures and their representations of technologies and technological choices solidifies the need to explicitly connect goals and requirement as given use cases properties to the selection of technologies. By directly referring to a BI reference architecture, which depicts the necessary building blocks to do this, an explicit and concise process can be formulated. In particular, the level of detail which the analyzed Big Data reference architectures employed to depict core functionality and data is of interest when making said connection, as these elements in the

reference architecture ultimately realize the goals and requirements from a given use case. This should be reflected by the solution.

SR_5 **The development process shall reduce the overall technological solution space when selecting technologies for a BI architecture.**

The BI solution landscape is more complex than the traditional technology solution landscape, which could already challenge project execution (e. g., through ill-defined requirements, cf. Section 2.5). Therefore, the development process should aim to reduce this solution space, by aligning specific technologies to input goals and requirements. The existing analysis of architectures in practice illustrated that specific needs lead to individual solutions. Thus, building on explicit goals and requirement should allow to select a subset of appropriate technologies for the given task. In essence, the goal of an actual technology *selection* should be achieved.

SR_6 **The development process shall be agnostic to system development approaches.**

To be independent of traditional approaches to requirements elicitation during system development as well as more modern agile methods (cf. Section 2.5), the solution should aim to work with goals and requirements that result from any of these, as long they contain the necessary information for the development process to continue. Specifically, a traditional specification of all requirements and goals before starting a project should be supported as well as agile methods, where only less information is defined beforehand and only amended or updated later on in the process.

SR_7 **The solution models shall utilize an appropriate modeling language and techniques.**

A language suitable to create models, e. g., Petri nets (cf. Section 2.5.4), which can comply with quality criteria such as the outlined GoM, can realize several benefits such as comparability and consistency.

4 Goal-oriented Business Intelligence Architectures (GOBIA) Method

In this chapter, the Goal-oriented Business Intelligence Architectures (GOBIA) method is described and illustrated using the FROG AIR case from Section 1.3 as running example. The GOBIA method addresses the solution artifact requirements defined in Section 3.7 and is the main contribution of this work. The name for the method is chosen to reflect the universal notion of BI for decision-making, independent of technology, and the alignment of BI architectures to specific goals by means of a development process.

First, an overview of the approach and its inception is given in Section 4.1. It explains the development approach to the method and positions it in an organizational context is, where its iterative execution is detailed.

Following this, the two main artifacts of the GOBIA method are detailed. The first artifact is GOBIA.REF in Section 4.2, which contains the structural elements for constructing customized BI architectures. In fact, two reference architecture models are employed to represent higher and lower levels of abstraction of a universal BI reference architecture. Next, GOBIA.DEV is explained in Section 4.3. GOBIA.DEV is the process that guides the customization of GOBIA.REF using goals and scenarios from a given BI use case. The result of this development process in GOBIA.DEV are concrete, customized BI architectures, which are viable alternatives for fulfilling the given use case requirements. By applying a three-phased refinement, functional and non-functional requirements are mapped into increasingly technical elements, which ultimately leads to a criteria-based selection of appropriate

technologies and tools, which in turn form the customized BI architectures. This guidance by means of models and a process providing a sequence of construction steps solidifies the constitution of GOBIA as a *method*.

In a post-GOBIA phase, which is outlined in Section 4.4, a detailed evaluation of these viable BI architecture alternatives takes place through typical organizational processes. Here, the approach for an evaluation using the perspectives of the TORE framework is elaborated upon. Lastly, methods to extend the GOBIA method itself are discussed in Section 4.5.

4.1 Overview of the Approach

In general, the GOBIA method features a development process *GOBIA.DEV*, which embeds two types of reference architectures summarized as *GOBIA.REF*. GOBIA.DEV is using BI goals and requirements of a specific use case as input. This ensures a goal-orientation of the approach and the result. The final output is a set of concrete, customized BI architectures, which consist of specific tools that are compatible with each other and can address aforementioned requirements. In summary, the GOBIA method consists of the following artifacts, which are further elaborated in this chapter:

GOBIA.REF represents the notion of a universal BI reference architecture in two different levels of abstraction and is pictured in Section 4.2.

> **A Functional Reference Architecture** offers a technology-agnostic, functional view on a higher level of abstraction with respect to abstraction levels of reference architectures in general. This view of the functional reference architecture includes typical data types and BI functionalities, especially descriptive and predictive analytics functionalities, typically found in BI systems.
>
> **A Technological Reference Architecture** depicts a layered structure of technology classes and is designed as representative overview of notable traditional and novel technologies, structured into aforemen-

tioned classes. This technological view is on a lower level of abstraction than the functional reference architecture described above.

GOBIA.DEV is a three-phase development process that results in a set of viable BI architectures. Its rationale and description are depicted in Section 4.3. A high-level overview of this flow is pictured in Figure 4.4.

Phase I uses the GOBIA.REF functional reference architecture to map use case requirements into BI functionality, high-level data preparations steps and conceptual data entities. The result is an *abstract BI architecture*.

Phase II ingrains architectural patterns by adding technical details regarding storage into the output of Phase I. This is achieved by mapping the three functional element types of the GOBIA.REF functional reference architecture onto technical Architecture Building Blocks. These are used to compile a *semi-concrete BI architecture*.

Phase III employs the GOBIA.REF technological reference architecture to map the intermediate, semi-concrete Architecture Building Blocks (ABBs) onto technology layers of aforementioned reference architecture. By providing decision criteria as process, Phase III guides the selection technologies that fit the case characteristics in form of functionalities and data as well as its technological requirements. This results in a set of compatible technologies from each layer for each ABBs, which are compiled into *customized BI architecture alternatives*.

In the following, the general design approach of the GOBIA method, rooted in method engineering, is detailed in Section 4.1.1. Next, Section 4.1.2 illustrates the scope of the GOBIA method within the organizational development and implementation of a BI architecture. Afterwards, Section 4.1.3 details the iterative nature of the GOBIA method.

4.1.1 Design Approach

The field of *method engineering* is dedicated to the study and construction of *methods* "for the development of information systems" [Bri96, p. 276]. To construct the GOBIA *method*, it is appropriate to apply a *method engineering* approach, which defines the framework that is used to define the context and basic design principles of the artifacts of the GOBIA method. In addition, method engineering is also tasked to define techniques and tools [Bri96, p. 276].

Mayer et al. documented their findings on method formulation and engineering, which emerged during several iterations of constructing their *Integration Definition (IDEF)* methods [MCF+95]. Extending upon the generic method definition by [MS95] in Section 1.2, Mayer et al. state that methods are synthesized from *best practices* in a method's domain [MCF+95, p. xi]. Their goal is to "facilitate a scientific approach to problem solving" [MCF+95, p. xii]. According to them, a method can be more formally characterized through the following aspects [MCF+95, pp. xif.]:

> **Definition.** The definition of a method consists of (1) the motivation for the method, (2) a set of underlying concepts and (3) the formal and informal approaches to represent the method [MCF+95, pp. xif.]. For the GOBIA method, the motivation was outlined throughout the previous chapters. Its goal is to guide in the process of formulating a customized BI architecture to be implemented based on given use case goals and requirements. Its target audience are practitioners in need of a BI system for decision making. The technical background on technologies in Chapter 2 and the analysis of analytical architectures in Chapter 3 should help to include best practices from a holistic Business Intelligence domain (cf. Section 3.1) into the GOBIA method.
>
> **Discipline.** This component encompasses the formal structure of the method and actual steps (i. e., the "procedure") to apply it [MCF+95, p. xi]. In essence, this part is the exposed interface of the method, which is presented to its users and by which these utilize it. Therefore, it contains visual models, such as architectures, or textual descriptions,

describing and illustrating the method's application. For the GOBIA method, these are the architecture and process models as well as textual descriptions depicted in Section 4.2 and Section 4.3. Oinas-Kukkonen postulates that, besides the construction of method artifacts, these models have to be argued for. He proposes that a method *rationale*, which describes the reason for a particular design of a artifact, should align usage of a method to its designed artifacts throughout its use [BLW96, pp. 87f.]. Therefore, aforementioned sections will also argue the rationale behind the artifacts of the GOBIA method. For instance, Section 4.2 explains the specific reasoning behind the structure of the proposed reference architectures. Mayer et al. propose an approach to developing methods [MCF+95, p. 7]. However, it resembles a typical design science approach, which is applied in this work (cf. Section 1.2), but is amended by the characteristic aspects described here. Thus, no deviations from the proposed design science method are necessary.

Uses. The third important aspect is a method's embedding into the context of an organization, where typically a plethora of different methods are applied. Here, its interaction with other methods or its standalone use is to be explained as well [MCF+95, p. xii]. This contextual embedding is elucidated in Section 4.1.2, where its positioning in an organization and its overall scope is illustrated, and in Section 4.1.3, where its execution in an organizational context is outlined.

Approach to Method Application. Brinkkemper et al. outline three categories for methodological approaches [BLW96, pp. 1f.]: *Flexibility*, *Controlled Flexibility*, and *Control*. While approaches in the *flexibility* category feature an "ad-hoc development" and offer only "few guidelines", approaches in the *control* category rely on "rigid guidelines" and require compliance of all results to a measurable approach [BLW96, p. 2]. The GOBIA method is designed as a *controlled flexibility* method. Such approaches offer uniform building blocks, which are adapted to specific project contingencies. Moreover, guidance for each building block is offered [BLW96, p. 2]. The GOBIA method proposes certain "paths", which can be selected (cf. [BLW96, p. 2]). These paths include architectural patterns, which permeate building blocks

from respective reference architectures. Conclusively, a flexible approach using a set of predefined building blocks is provided. As Mayer et al. note, a method and its architectures focus on selected criteria and characteristics during construction. They are not designed to capture every conceivable circumstance, as this would ultimately negate the need for a method (cf. [MCF+95, p. xiii]). Similarly, the GOBIA method is focused on the design of customized BI architectures, which are built using specific predefined technologies and technology classes. For example, custom implemented software applications are by design not captured by the GOBIA method. This applies to any tool not covered by the confines of its structure. Apart from that, extension aspects of the method are handled in Section 4.5. To work around the disadvantage of predefined building blocks and certain paths, the GOBIA method aims to ensure that at least the identified architectural patterns can be covered. The designed method artifacts, both visual and textual, are part of a so-called *method base* (cf. [BLW96, p. 191]). The descriptions of GOBIA.REF and GOBIA.DEV as well as the supplementary data in Appendix B and Appendix C, constitute this method base.

Method Design Language. GOBIA.DEV models are primarily visualized using *Petri nets* (cf. Section 2.4), which are accompanied by the textual descriptions in the following sections. The final GOBIA.REF models are embedded into the GOBIA.DEV process models so that both possess a formal Petri net representation. In general, Petri nets as process modeling language are an appropriate fit to formally represent a method's "list of steps" [MS95, p. 276]. Moreover, they feature only two design elements (places and transitions), which allow for a more streamlined visualization of processes. In addition to that, tokens inside the places allow to carry information and results of the method as data objects, which are manipulated and refined by the method. Figure 2.9 in Section 2.4 highlights certain visual customizations for Petri nets that aid in modeling visually less-complex models, i. e., with fewer overall elements. For added consistency, the resulting architecture models are visualized in a Petri net notation as well, but structured in such a way that they represent an architecture or other intermediate results. It should be noted that the separate depiction of the rationale of the GOBIA.REF model in

Section 4.2 is illustrated in a visually simpler format using only blocks that resemble Petri net transitions, as the focus there is to explain the specific structure of these reference architecture models.

4.1.2 Method Scope

The proposed development process GOBIA.DEV does not stand separately in an organization. It is embedded into a greater organizational context, which eventually triggers the development of a customized BI architecture. Notably, the customized architecture is eventually implemented outside the GOBIA context. The evaluation and final selection of the feasible BI architecture alternatives determined by the GOBIA method is realized as typical software selection process in an organizational context under consideration of detailed technological, organizational, economical, and regulatory (TORE) factors. This is discussed in Section 4.4. In addition to an implementation, the use and constant monitoring of the implemented system is conducted in the organizational context outside the scope of GOBIA method. The surrounding context is depicted conceptually in Figure 4.1. Activities in this context are visually structured according to their roles using the perspectives of the TORE framework (cf. Section 2.6.2) amended with an external perspective on the market.

This greater context ultimately leads to the definition of the relevant goals and requirements as important initial input for GOBIA.DEV. Through its phases, GOBIA.DEV compares intermediary results to these goals and requirements so that corrections can be applied to the results. Additionally, the goals and requirements can be updated as well. Hence, it is important to specify the necessary information and their level of detail of goal and requirement definitions. This is detailed in the description of each GOBIA.-DEV phase in Section 4.3.

External and Internal Change Impetus. To start an iteration of the GOBIA.DEV process, an internal or external *change impetus* to the company is assumed. A change impetus is a trigger that is activated when internal or external conditions necessitate an organization to adapt. Such adaptations

can ultimately lead to more or less greater changes in the underlying BI architecture, as decision making needs to reflect these needed changes as well. When such a trigger is fired, new strategic goals are set or existing ones are updated, respectively superseded. The distinction into external and internal triggers to GOBIA is derived from the typical strategic planning viewpoints on an organization (cf. Section 2.6.1). Here, an organization can act according to external market forces (e. g., Porters Five Forces [Por79]) such as new competitors or regulatory changes. Acting on these forces is also called strategic positioning in the *Market-based View (MBV)* (cf. [McG15] [Ros03, pp. 83ff.]). When a strategic position is to be shifted, the organization needs to align to that positional shift. Thus, a BI architecture should be aligned to that strategy. The actual reformulation and adaption process can be supported by the GOBIA method. For instance, if a company enters a new market with a new product, extended information on the prospective customers may be necessary. The organization must then evaluate and adapt its analytic architecture to source and analyze appropriate information. On the other hand, internal triggers might set off a need to change, too. In a *Resource-based View (RBV)* (cf. [MP92] [Ros03, pp. 86ff.]), an organization's own capabilities can be a force to gain a (sustained) competitive advantage [PI05], which again would influence goals for BI architectures as well.

Set or Update Goals. An activity to realign a business and update or set respective business goals can be triggered by three activities. Besides the aforementioned external and internal change impetus, the *Use and Monitor Architecture* activity can yield that a current BI architecture is not aligned to its existing goals any longer or that lessons learned from using such architecture in practice lead to the desire to change it by updating its underlying goals. First, high-level strategic goals might be formulated. However, eventually, more specific goals are needed, which allow the elicitation of BI requirements (cf. Section 2.5.1) for a BI architecture in the next activity. The organizational processes, which are related to implement strategies into more a high-level business goals, which lead to specific goals for BI system are not further studied here and out of scope for this work. However, several approaches are available for further reference on this topic such as

Strategic Information Systems Planning (SISP), which is an actively researched approach to establish alignment between strategies, business goals, and information systems (e. g., cf. [SGT98, Teu07, TP13]).

Derive BI goals and requirements. After goals are set or updated, this activity captures initial requirements specific to a BI system, which are amended with initial goals. Hence, the term *BI requirements* is used. Notably, the GOBIA method can explicitly work with *initial* goals and requirements, where not all details are fully specified. Limiting this activity to traditional approaches, where a complete system is fully specified before further planning and implementations begins, would exclude an agile and lean approach to system development. Hence, one principle of GOBIA is to work with the minimum amount of information as it is needed in each of its phases. This ensures that GOBIA can be executed regardless of the development approach chosen. Moreover, the term *initial* is to be understood in a sense that the GOBIA method can lead to refinements or changes of these requirements by adapting the goals of the system. With respect to the discussion regarding known business questions or data in advance in Section 3.2, GOBIA assumes that initial characteristics of business question and data are known. Exploratory use cases can be reformulated as distinct business questions to be applicable for GOBIA.

GOBIA Method and Extended Scope. The scope of the GOBIA method starts with the initial BI goals and requirements as input values. It employs the three-phased GOBIA.DEV development process (cf. Section 4.3) to customize the reference architectures given by GOBIA.REF (cf. Section 4.2) into viable and concrete BI architecture alternatives. The extended scope of the method starts after GOBIA.DEV has concluded its third phase. Here, organizational activities are conducted to evaluate and assess the outputs of GOBIA.DEV and select a final BI architecture to be implemented afterwards. An overview of these activities is presented in Section 4.4.

Implement Architecture. The implementation of a BI architecture involves several activities, that are out of scope of this work. For instance, this includes procurement of hardware and software for the architecture as well

as building data pipelines, statistical or database models, and connection to possible usage endpoints (e. g., single computers or visualization tools). Notably, the value chain activity of data usage, which is only broadly covered in the core scope of GOBIA and where the aforementioned usage endpoints refer to, requires additional decisions before implementation. Depending on technology and methods to be used, different approaches are applied. For instance, several literature sources exist to determine dimensions and data sources for a DWH (e. g., [KC04, KR13, Inm05]). Moreover, the designed BI system might require integration with other systems, e. g., if a combined OLTP and OLAP usage is envisioned.

Use and Monitor Architecture. In the usage phase, the implemented BI architecture is used for its designated purpose in the greater context of providing decision support (cf. Section 2.1). Here, monitoring the use either quantitatively or qualitative yields new input for changes to an already running BI architecture. For example, quality of mathematical models can be validated using appropriate statistical measures. Apart from that, latency and throughput can be monitored to verify if real-time or alike requirements are met. Qualitative methods include studying behaviors and opinions of users (e. g., using surveys, cf. [SG07]).

Re-Evaluate Goals. During each phase of GOBIA.DEV, it is possible to determine that the process cannot continue and that the input goals as most high-level input need to be refined and reevaluated. This can lead to an updated set of goals. The process can continue at the previously exited phase or at any previous phase, because a re-evaluation of goals can also lead to updated interpretations and refinements of requirements. In case the goals cannot be refined and yet a GOBIA.DEV phase cannot be completed as there is a conflict in achieving these goals (e. g., the goals are too ambitious for the represented technology or generally unfeasible) the process is aborted.

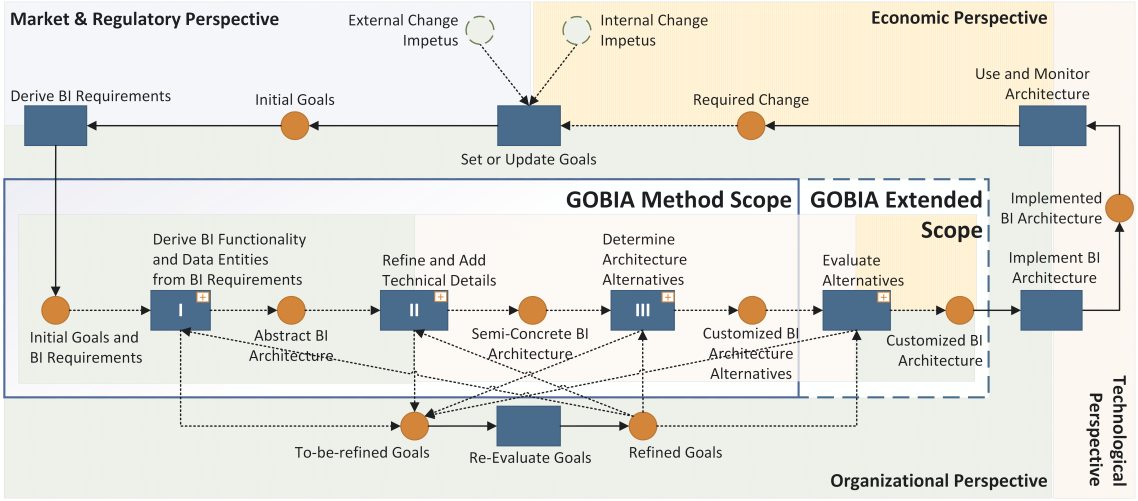

Figure 4.1: Execution of the GOBIA method illustrated in its own scope as well in an extended general GOBIA scope. Depicted as Petri net process. Four perspectives according to the TORE framework and external market consideration are visualized to denote in which realm an activity is primarily executed.

4.1.3 Execution in Cycles

In contrast to static development approaches, such as the Waterfall model in software engineering, the GOBIA method is designated to be integrated into an iterative process. Such iterative business practices can be represented by in the so-called *Deming cycle*, which was proposed by Deming, which is rooted prior work from Shewhart and other from the first half of the 20th century (cf. [Dem86, pp. 88f.]). It defines four phases for planning, executing, monitoring, and reacting [Dem86, p. 88]. More common names of these phases are "plan", "do", "study" (alternatively: "check"), "act" (PDSA), whereas several alterations and slightly different interpretations exist (cf. [Moe09, MN10]). These phases form a circle. After the fourth activity concludes, Deming postulates that a repeated execution of a cycle is done with accumulated knowledge [Dem86, p. 88]. In general, its usage is designated for high-level management tasks, e. g., for introducing and improving products, but new uses in quality control emerged as well [MN10]. The organizational context and execution of the GOBIA method resembles the basic principle of this Deming cycle. In the following, its four phases are outlined and related to the context of the GOBIA method (cf. Figure 4.1).

Plan. The activity of setting up a plan aims to set a *goal* that should be achieved. Simultaneously, "desirable changes" and available data are assessed. If information is intensification, further monitoring might be necessary [Dem86, p. 88] [MN10]. For GOBIA, it is assumed that the elicitation of business goals is at least started before starting the GOBIA method. While a specific refinement of BI goals and requirements is not in immediate scope of GOBIA, it can be done in parallel as long as needed information can be supplied.

Do. Next, the objectives are realized by *executing* the set-out plan. In the context of GOBIA, this phase follows after GOBIA.DEV has concluded and the final BI architecture has been determined in the extended GOBIA scope.

Study. The study phase *monitors* and investigates the effects of the implementation. In case of BI architectures, this could include performance metrics (e. g., latency or throughput) or user satisfaction surveys. [MN10]

Act. After monitoring the effect, learnings and initial changes can be identified [Dem86, p. 88]. Also, as *reaction*, it can be decided if and when the next phase should be executed [MN10]. These learnings drive the planning process of the *next cycle iteration*. In the GOBIA context this would mean triggering an internal change impetus, as usage of the implemented BI architecture reveals that changes are required ("learning"), which lead to an update of respective business goals.

It becomes clear that the GOBIA method is located between the planning and the executing phase, while aiding the completion and refinement of the latter. This underlines the general approach of the GOBIA method, which should be able to start with a minimum set of information. This makes it compatible to both traditional approaches, but also agile approaches. Importantly, the GOBIA method builds on continuous improvements. In summary, the goal of the GOBIA method with this approach is to dynamically react to changes, as soon as business goals are updated.

4.2 GOBIA.REF

GOBIA.REF denotes the notion of a unified BI reference architecture that encompasses both traditional and novel technologies for the purpose of Business Intelligence. A technology map with typical roles or any form of static architecture that include the complete set of all involved technologies is infeasible on its own nowadays. For example, a technology map contains typical architectural layers and possible technology classes (e. g., NoSQL data stores) or specific technology choices (e. g., document stores). A superset of all possible architectural patterns as described in the previous section would connect all these elements in a multitude of ways. To be able to construct a customized architecture, more guidance is necessary — as postulated in the solution artifact requirements in Section 3.7.2. To make a proper selection,

both a development process (cf. Section 4.3) and additional information for selection are required. The latter can be gathered from the goals and requirements.

In total, GOBIA.REF encompasses two reference architecture models. In the following, these are illustrated separately and elaborated upon. When discussing the models separately, the focus lies on depicting their structure and organization and explaining the rationale behind that. Thus — for the sake of discussion — Petri nets are depicted in a simplified form throughout the sections involving GOBIA. However, their gestalt is visually consistent to their embedded forms inside GOBIA.DEV. Additionally, their dynamic use for architecting and building relationships between models including the selection as well as refinement of elements in GOBIA.DEV is discussed in Section 4.3.

4.2.1 Functional Reference Architecture

The GOBIA.REF functional reference architecture, depicted in Figure 4.2, is the entry point for the development of a customized BI architecture. It represents the use case side of a BI system and the highest level of abstraction on the journey from reference architectures to specific architectures as outlined in Section 3.1. Instead of focusing on technologies or tools, it focuses on abstract functionalities and data, possibly connected by abstract data preparation activities. These should help to bridge the step from goals and requirements to technology selection. This is the case, because functional requirements directly relate to functionalities a system should exhibit (cf. Section 2.5.2). In case of BI projects this is ultimately often analytics functionality, as it is the final step in generating value and insights from data as demonstrated with Big Data value chains in Section 3.2. Insights are the basis for a variety of data uses by both humans and machines for, e. g., as visualized graph on a management dashboard. As indicated in Section 4.1.2, the scope of the GOBIA method with respect to value chain activities is focused on the part between data generation until briefly after data analysis, which is why the large variety of data usages are out of scope.

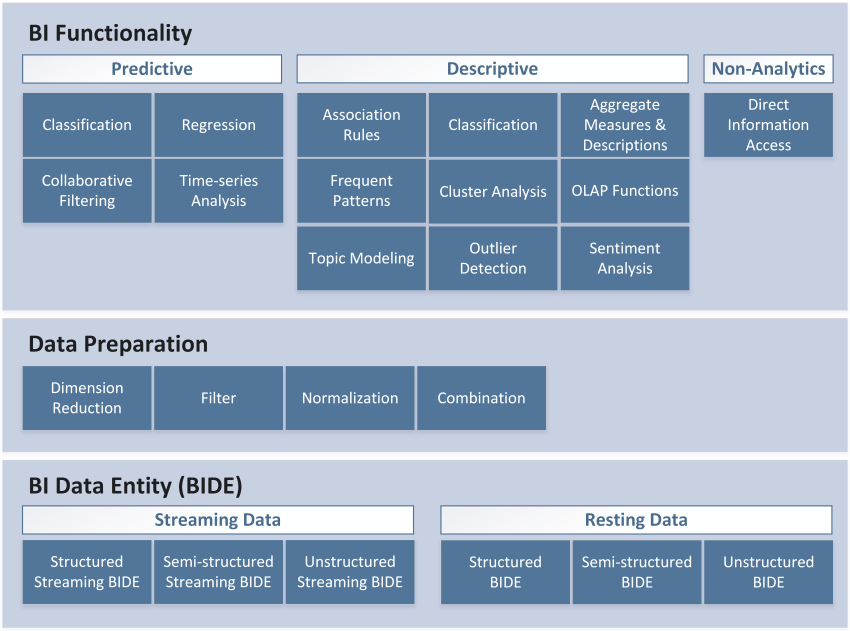

Figure 4.2: GOBIA.REF Technological Reference Architecture consisting of BI functionality, data preparation, and BI data entity elements.

In general, functionalities and data in the functional architecture are represented as high-level elements and in an abstract fashion. For example, a *Classification* functionality block is placed in this first reference architecture instead of immediately listing all possible sub-methods and algorithms. The guiding principle is that only as the minimal set of information required should be specified in this first architecture. During GOBIA.DEV, initial elements from the functional architecture are further refined, where necessary. The overall goal is to only gather the minimally required information for technology selection at the end of GOBIA.DEV.

The functional reference architecture defines three types of abstract elements structured in three corresponding layers:

BI Functionality depicts technology-independent core BI functionalities, which are focused on analytics and information provisioning. Functionalities are grouped into three distinct categories: predictive analytics, descriptive analytics, and non-analytics. The first two are derived from the three analytics types outlined in Section 2.3.1. For the functional reference architecture model, both prescriptive analytics and predictive analytics are represented by a predictive functionality block, because both technically build on prediction as core functionality. Non-analytics is added to complete the coverage of BI use cases. In some BI use cases it is sufficient that previously integrated data is simply presented to users, without applying any analytics. For example, it might be sufficient to show a table with integrated and cleansed data without any further aggregations or other descriptive measures.

For each of the three BI functionality categories, several high-level functionalities are defined, which are based on the analytics methods outlined in Section 2.3.3. All but one typical method are represented by BI functionalities, whereas the final method is covered by a data preparation activity below.

Predictive contains two main functionality blocks for *Classification* and *Regression*, which discriminate all typical methods and algorithms for predictive analytics. Additionally, two more specific methods are added as quickly accessible functionality blocks. *Collaborative Filtering* represents a popular predictive classification method for recommender systems, while *Time-series Analysis* is a widely used predictive regression method.

Descriptive represents a wide range of descriptive analytics methods. *Association Rules* and *Frequent Patterns* cover the field of pattern, association, and general correlation mining. *Cluster Analysis* and *Classification* are method classes for grouping similar data. Notably, the latter can also be used descriptively to label existing data. *Outlier Detection* represents

methods to find anomalies in data, which can be utilized separately or serve as input for other methods. *Aggregate Measures & Descriptions* comprises all aggregate measures to characterize and discriminate data. This includes finding maximum and minimum values, averages, counting functions, distributions, and variances. Naturally, these can be applied to all data or subsets of it. The large group of *OLAP Functions* covers complex functionalities and operators found in OLAP applications. Here, cube operators work on multi-dimensional data models and allow for, e. g., drill-down, roll-up, slicing and dicing, and pivot operations [HKP11, pp. 147f.]. Typically, such functions are applied as interactive analyses on multi-dimensional data models. Notably, OLAP often offers capabilities to compute aforementioned aggregate measures and descriptions, as OLAP operators themselves depend heavily on them. However, aggregate measures are still separated as distinct functionality block, since not all complex OLAP functionalities might be needed. Lastly, the descriptive blocks are augmented with *Topic Modeling* and *Sentiment Analysis*, because they represent notable methods in the Big Data field and its new domain of text analytics that should be directly accessible as functionality blocks (cf. Section 2.3.4).

Non-Analytics consists of one functionality block for *Direct Information Access*. As sole functionality not related to analytics, it covers use cases, where relaying integrated data to a user without analytics is needed. In this case, the value of data is generated solely by cleansing and integrating it. Accessing information can be conducted, e. g., by displaying a raw data table or respond to simple data query. However, it should be aligned to the purpose of BI, which is to provide information for decision making. This functionality is founded in notion of "data access and retrieval tools" in the context of information processing in Data Warehouses by Han et al. [HKP11, p. 153]. Here, this functionality is to be understood in a general sense, unrelated to specific technology. However, simply relaying data and enabling information access might warrant to simply re-use technology choices for preceding BI functionality (cf. Section 4.3.2).

Data Preparation denotes optional high-level activities to prepare data sources before processing with BI functionality. These conceptual activities should be distinguished from fine-grained data pre-processing activities in an ETL process, such as data integration and cleansing, which will be introduced in GOBIA.DEV. The high-level activities here are more coarse-trained tasks and omit technical details. For instance, a format transformation task could be a technical necessity due to the combination of two data sources to implement an analytics functionality. *Dimension Reduction* techniques are used to remove unnecessary dimensions from data. This is particularly important for high-dimensional Big Data with many attributes. *Filter* activities encompass removal of unnecessary data points (e. g., rows). This can include complex statistical sampling techniques of a subset of data, if necessary. Other means are simpler filters as found in data pipelines of streaming processing systems, where simple criteria are used to include or exclude data. *Normalization* refers to a statistical normalization values, e. g., ensuring that units of measurements are equal or absolute and relative formats are harmonized. Lastly, *Combine* subsumes typical merge and join activities, when two or more data sets should be combined or integrated, before a BI functionality or other preparation activities can consume them. Nevertheless, this activity should be placed when the combination of sources has a more permanent nature.

BI data entity represents conceptualizations of data sources, which are to be used in a BI system according to the requirements. BIDEs abstract from specific data storage technologies such as RDBMSs, because these are not important in this phase. For instance, instead of physical sources such as "ERP RDBMS" a specific BIDE "Inventory data" or "ERP data" can be formulated during GOBIA.DEV Phase I so that the focus is on the required contents, which are consumed by the BI system. However, it might also be feasible two have more than BIDE per physical source to distinguish data by its needed contents. BIDEs are categorized by data velocity first and the three basic forms of data variety, i. e., data structure, second. Velocity and variety are the first key determinants later on, which is why they are focused here. With regard to velocity, BIDEs are classified either as resting data or as

streaming data. Each BIDE can either be a structured, semi-structured, or unstructured data entity. Notably, a desired data structure can be defined for BIDE. For example, if a requirement necessitates structured tabular data, a BIDE can be defined accordingly. Later, differences between actual data sources and BIDEs are resolved through acquisition, processing, and storage elements.

4.2.2 Technological Reference Architecture

The technological reference architecture (see Figure 4.3) in GOBIA.REF structures technologies into classes, which were primarily elucidated in Chapter 2. The technologies are assigned to one of five layers according to their typical roles within the layers. For each technology inside each layer, several product alternatives (*tools*) are available, which become part of the final result through GOBIA.DEV. Inside each layer, technologies are grouped into certain technology classes, which are selected first (e. g., graph databases as technology (subclass) belong to NoSQL data stores as technology class inside a data storage layer). Figure 4.3 additionally color-codes technologies as traditional, novel, or a mixture of both (e. g., if a traditional technology has been extended with novel approaches). For instance, semi-structured data was only partly utilized by traditional approaches, but gained traction with novel ones or distributed RDBMS, which are derived from traditional technology (cf. Section 2.2.7).

A selection of representative exemplary tools is provided for each technology class in each layer to provide alternatives for initial tool selection. This categorized list of tools is termed the **GOBIA tool repository** and is utilized in GOBIA.DEV. To this end, a brief tool overview is provided to identify representative tools — unless the respective sections of Chapter 2 already provided such tool lists. The process of extending the repository with new tools is discussed in Section 4.5. The detailed tool lists for each layer, including key characteristics and supplementary technology data, can be found in Appendix C.

The composition of layers is based on the proposed unified Big Data value chain from Section 3.2.2. This layout covers both novel and traditional architectures. The analysis for reference and actual architecture in the previous

chapter affirmed that this synthesized view on a unified architecture, grounded in literature, can be an underlying foundation for analytical architectures.

Thus, this value chain is used as foundation for constructing layers of GOBIA.REF. This allows to organize technologies by typical roles and activities. Importantly, while such structural aspects of a map organization are detailed, the actual factors for selection those tools are detailed in the subsequent Section 4.3.

For deriving the GOBIA.REF layers based on value chain activities, these have to be re-customized to their new purpose. Primarily, the value chain activity "Data Acquisition and Processing" is divided into two distinct layers: one for data acquisition and the other for data processing. The underlying rationale is a separation of concerns. Particularly, processing functionality can be contributed to by tools also suitable for certain acquisition tasks, e. g., stream processing frameworks. Additionally, both DBMSs and NoSQL data stores offer in-database processing functionality, e. g., SQL processing. A third possibility is dedicated processing environments such as MapReduce. To accommodate this situation, a processing layer is considered separately. Moreover, because of this separation, dedicated technology and tools for data acquisition such as ETL tools can also be detailed distinctly.

Data Generation

This special layer is concerned with data characteristics, which provide key properties to drive technology selection in the other layers in GOBIA.DEV.

In conformance to the previously defined BIDEs, data is structured along the dimensions *velocity* and *variety*. However, instead of conceptual BIDEs, data here refers to actual data from physical data sources. Streaming and resting data are distinguished. Unless noted otherwise, "data" refers to resting data, while streaming data is explicated. Notably, resting data can have a certain velocity to it, without being streaming data. Technology choices in GOBIA.DEV are made accordingly (e. g., micro-batches, cf. Section 2.2.6). With respect to variety, unstructured, semi-structured, and structured data is distinguished. A combination of these could yield, e. g., *unstructured streaming data* such as a video stream.

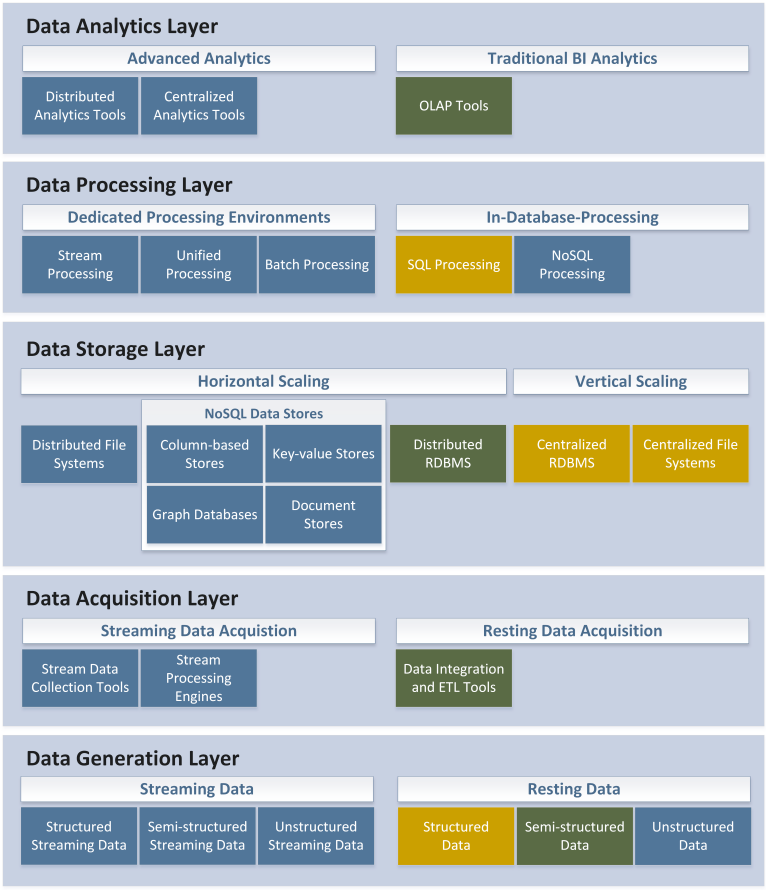

Figure 4.3: The GOBIA.REF Technological Reference Architecture structures various technology into technology classes and layers based on a unified Big Data value chain proposed in Section 3.2.2. Technology is represented by various software tools. Tools are complemented by approaches to processing. Additionally, incoming data is disseminated according to velocity and variety dimensions. Traditional elements are color-coded in light-*yellow*, novel ones in dark-*green*, and mixed ones in *blue*.

The analysis of contemporary architectures and of Big Data in general clarified that the velocity and variety dimensions are the key characteristics of data.

Data Acquisition

Acquisition of data is tasked with making incoming data available to the BI system. Technology wise, this is achieved through data ingestion. Several tools are available for this, of which a representative set for inclusion in the GOBIA tool repository is detailed in the following.

A key distinction characteristic for acquisition tools is data **velocity** and typically associated to latency requirements. The reasoning is that resting data can be ingested in batches with several degrees of pre-processing (e. g., cf. Section 3.3), while streaming data requires a different approach, which involves, e. g., continuous computations and data flow management (cf. Section 2.2.6). The tools that are initial part of the GOBIA tool repository for acquisition tools are listed in Table 4.1.

Table 4.1: GOBIA tool repository overview for the Data Acquisition layer. The full list including sources and properties is located in Appendix C.

Technology Class	Tool
ETL and Data Integration Tools	Pentaho Data Integration (Kettle) Talend Open Studio Apache Sqoop
Streaming Data Collection Tools	Apache Kafka Apache Flume Apache NiFi
Stream Processing Engines	Apache Spark[xvii] Apache Storm Apache Flink Apache Kafka Streams

[xvii] With Spark Streaming.

ETL and Data Integration Tools. Typical *ETL tools* are used for extracting, cleansing, integrating, and loading data into a target storage, often RDBMSs (cf. Section 2.1.2). Strictly speaking, extraction fulfills the function of acquiring data. These tools also transform various incoming data into a structured or another output format. They are traditionally used for batched-based ingestion, which is regularly executed but not with real-time or near real-time latencies (cf. Table 2.5). ETL tools are a large part of a greater data integration tools market. Gartner asserts that a current trend is that more capabilities are subsumed under the umbrella of *data integration tools*. Thus, tools marketed as data integration tools are suitable here as long as they possess necessary ETL capabilities [178]. As this category is mature, numerous tools are on the market, among them several commercial software suites and appliances (e. g., cf. [186]). When used for a Data Warehouse, ETL processes are complex and resource-intensive to attain necessary integration and quality levels. Notably, ELT and ETLT variations of ETL are also possible. Some ETL tools can be deployed in a distributed environment, but their horizontal scalability is inhibited by their architecture and approach, especially when they use an RDBMS or other technology, which does not scale linearly. To represent traditional ETL tools, *Talend Open Studio* and *Pentaho Data Integration (Kettle)* are selected. Although specific features between these and other commercial vendors differ, these technology class representatives are sufficient to disambiguate this technology class from the other two classes, because their typical functionalities (e. g., data integration and cleansing) and mode of operation (batch-based ingestion) differ. *Apache Sqoop* (cf. Section 2.2.5) is added as third tool option, because it represents the Hadoop ecosystem variant for loading structured data sources into HDFS and other Hadoop-related tools. Importantly, it does not feature a graphical user interface[1] and runs distributed via MapReduce tasks. Thus, it is a horizontally scalable alternative to the centralized tools.

Streaming Data Collection Tools and Processing Engines. For streaming data acquisition, two categories can be distinguished based on Section 2.2.6. Most of these systems are inherently horizontally scalable and work in a

[1] This applies to version 1.4.7 recommended for productive usage [99].

distributed fashion. They are designed for low latency environments. Due to this, they can offer high data throughputs, making them suitable to ingest and process high data volumes over time. Tools may offer support to not only work on streaming data exclusively, but also support more traditional sources such as Web logs (e. g., Apache Flume), or structured streaming with declarative query language support or even a unified processing model for either batch or stream processing (e. g., Apache Spark Streaming). Typical stream collection and processing applications work differently than applications for resting data [Psa17, pp. 61f.]. ETL tools prepare data for subsequent storage and ad-hoc queries that are answered based on prepared resting data. Stream collection and processing systems work with data "on-the-fly". They set up a query, which is represented through a topology of processing elements such as a graph. Then, incoming data flows "through" [Psa17, p. 61] this topology and is transformed — depending on the stream processing capabilities — and relayed to a data sink.

Streaming Data Collection Tools. These tools are concerned with collecting streaming data and providing data flow management. The latter includes message queuing to avoid congestion as well as mechanisms for fault-tolerance, reliability, and availability (cf. Section 2.2.6). Furthermore, these tools usually act in a distributed instead of a centralized manner. In general, only rather simple transformations are applied within their ingestion pipelines. For instance, Apache Flume or NiFi (cf. Section 2.2.6) are distributed tools for ingesting Big Data with low latencies and are especially suitable for streaming data. However, both can also apply some transformations while moving data and can be used to combine several (streaming) data sources. However, depending on the required latency, raw data or minimally transformed data can be passed down a streaming pipeline (i. e., low pre-processing intensity is exhibited). Apache Flume, NiFi, and Kafka are selected as tools for the tool repository. They represent typical streaming data collection tools.

Stream Processing Engines. Such tools are often part of a larger framework and offer more complex filtering and transformations and may even allow for analytics and machine learning (i. e., medium to

high pre-processing intensity). A typical use case for them is Complex Event Processing (CEP) [322]. Often these engines are part of larger frameworks (e. g., Apache Spark Streaming, Storm, or Flink), which can indeed independently access data (e. g., by tapping into streams via Internet sockets [97]) to ingest it into their pipelines. Stream data collection tools can especially provide reliable messaging and stream interaction patterns such as publish/subscribe mechanisms for stream processing frameworks. Apache Kafka is often regarded as such a data collection tool, which offers a basic streaming infrastructure and is used with Kafka Streams, Storm, and other aforementioned stream processing engines [284]. Conclusively, the aforementioned examples as dedicated stream processing frameworks (i. e., Apache Spark Streaming, Storm, and Flink) as well as Kafka Streams are used as initial selection for the GOBIA tool repository.

Data Storage

The data storage layer consists of storage technologies for persisting data. Two conclusions can be drawn from the inspection of the present storage technology landscape and respective usage patterns. First, any form of persistence in a modern BI architecture is optional and only depends on the specific use case. In a DWH reference architecture, Data Warehouses are a constant element where integrated data is stored. In novel Big Data scenarios, storage is more dynamic as the synthesis of a unified Big Data value chain in Section 3.2.2 indicated and the architecture analyses in Chapter 3 illustrated. For example, streaming applications may not have any form of primary storage to fulfill real-time requirements, while at the same time data lakes based on HDFS are promoted as common data landing area for "all data". Moreover, such a landing area can be placed in an architecture after several preprocessing and other smaller storage solutions. Second, the form of storage became more flexible. In addition to centralized systems, distributed storage with horizontal scaling allows to cope with large volumes of data. NewSQL RDBMSs add horizontal scalability as option for structured data storage. Additionally, NoSQL data stores introduced more

flexible data models to tackle high data variety and offer indexed storage as needed. Simultaneously, trade-offs between consistency, availability, and partition-tolerance can be chosen.

The technology and tool landscape for data storage can be structured by two dimensions. First, centralized storage systems that can only scale vertically are disambiguated from horizontally scalable systems (cf. Section 2.2.1). The next dimension refers to the primary storage model. Here, file-based storage is discerned from indexed storage systems. Indexed storage refers to a wide spectrum of systems from strictly structured relational systems, over still complex yet more schema-flexible systems such as document stores, to simplistic key-value stores. An overview of initial tools chosen for consideration in GOBIA.DEV is listed in Table 4.2.

Horizontal Scaling. Distributed storage systems possess the ability to scale out horizontally. *Distributed file systems* with HDFS as prominent representative allow storage of arbitrary files irrespective of data format and exhibit various advantages such as partition-tolerance and replication capabilities. The large group of *NoSQL data stores* consists of the four types of systems outlined in Section 2.2.2. Almost all of these stores have options for in-memory storage and different or even configurable CAP properties (e. g., *CP* or *AP*). Lastly, *distributed RDBMS* with SQL supported are enabled by several contributions from, e. g., NewSQL DBMSs, relational IMDBs, and horizontally scalable DWH (cf. Section 2.2.7). The following data storage representatives are selected for consideration in the GOBIA method. For distributed file systems, HDFS is chosen. For each NoSQL data store type two representatives are included. Redis as *CA, AP* capable in-memory solution and Riak KV as horizontally scalable *AP* system with regular disk support are alternatives for key-value stores. MongoDB and Couchbase represent two popular *AP* and *CP* scalable options for document stores. Moreover, Apache Cassandra and Apache HBase represent similar options for column-based stores. For graph databases, Neo4j is added. MySQL NDB Cluster and Cockroach DB are included for the class of distributed RDBMSs. The aim of this initial selection for the tool repository is to create a diverse, case-independent set of tools for the Data Storage layer, e. g., with respect to the CAP focus of

the considered tools. After GOBIA.DEV has concluded and a set of suitable storage tool alternatives was generated, benchmarks may need to drive the final storage tool decision for a specific implementation (cf. Section 4.4). Such evaluations may also benefit from a diverse tool repository.

Vertical Scaling. A wide range of mature traditional storage options with primarily vertical scalability are available. On the one side, there is a multitude of RDBMSs with strong ACID guarantees and SQL capabilities. For the GOBIA tool repository, MySQL and PostgreSQL are chosen as popular and thus representative examples (cf. [217]). Centralized file system storage means that a local disk, but also be a network drive or SAN is used. Such disks appear as locally mounted disks to an OS. For the actual file systems of these disks, numerous options are available such as FAT32, NTFS, ZFS, or AppleFS. These are not further differentiated here. A generic item *Centralized File System Access* is added to the tool repository, which collectively represents the aforementioned alternatives.

Table 4.2: GOBIA tool repository overview for the Data Storage layer. The full list including sources and properties is located in Appendix C.

Technology Class	Tool
Distributed File Systems	Hadoop Distributed File System
Key-value Stores	Redis
	Riak KV
Document Stores	MongoDB
	Couchbase
Column-based Stores	Apache Cassandra
	Apache HBase
Graph Databases	Neo4j
Distributed RDBMSs	MySQL NDB Cluster
	Cockroach DB
Centralized RDBMSs	MySQL
	PostgreSQL
Centralized File Systems	Centralized File System Access

Data Processing

The data processing layer includes several processing approaches, which denote transformations into the form required by subsequent activities, e. g., data analytics or data usage (cf. Section 3.2.2). Processing abilities are also contributed by several technologies and tools from other layers (cf. Table 4.3).

Dedicated Processing Environments is comprised of tools that have a specific purpose towards processing data instead offering or focusing on storage. Notably, most of these environments emerged in the Big Data age. Consequently, these environments build on *distributed processing*. Three different approaches can be disambiguated:

> **Stream Processing** is suitable to process streaming data. Tools dedicated to streaming data exclusively exhibit a native streaming data model, which is appropriate for real-time requirements. These adopt a specific architecture to avoid latencies, which would be introduced by, e. g., synchronous data storage.
>
> **Unified Processing** lets users select between modes for processing streaming data or resting data. The latter is supported by batch processing approaches. Such unified pipelines may need to compromise, e. g., with respect to strict real-time requirements. Apache Spark is a notable example, which offers in-memory processing of both batch data and streaming data, for which it uses a micro-batch model.
>
> **Batch Processing** approaches focus on high-volume resting data, which may also exhibit a large data variety. Hadoop MapReduce is a widely known representative of batch processing approaches. It does not feature the data store and is entirely focused on distributed data processing.

In-Database-Processing relies on processing capabilities offered by indexed storage technologies of the data storage layer, which are SQL-capable RDBMSs and NoSQL data stores with varying access and processing capabilities.

> **SQL Processing** describes processing using SQL as query language and leveraging its capabilities as DML. As illustrated in Section 2.1.1,

SQL offers several capabilities to calculate aggregate measures, such as averages or minimum and maximum values. Founded on the relational model, sophisticated transformations and queries, particularly involving joins, can be formulated using SQL. The maturity and widespread use of the language and its declarative nature contribute to its popularity (cf. [217, 324]). Besides RDBMSs, Apache Hive, SparkSQL and other similar tools strive to offer so-called SQL-on-Hadoop and other SQL-like functionality.

NoSQL Processing is characterized by a wide range of functionalities and methods. In contrast to SQL, there is no common language for NoSQL processing. Sophisticated variants of it include an SQL-like query language, which has several limitations due to the nature of NoSQL data stores. Apart from that, document stores and column-based stores can allow for more sophisticated functions, which include filtering, index-based searching, and even aggregation (e. g., in MongoDB, cf. Section 2.2.2). Especially key-value stores may only offer simple retrieval methods, e. g., using a REST API, where a key is submitted and a corresponding value is returned or updated.

Table 4.3: GOBIA tool repository for the Data Processing layer.

Technology Class	Tool
Stream Processing	Apache Storm
	Apache Flink
	Apache Kafka Streams
Unified Processing	Apache Spark
Batch Processing	Apache Hadoop MapReduce
	Apache Flume
SQL Processing	*See Centralized and Distributed RDBMSs in Table 4.2*
NoSQL Processing	*See NoSQL data stores in Table 4.2*

Data Analytics

The field of data analytics can be categorized into the technical fields descriptive and predictive analytics as done for the functional GOBIA.REF model in Section 4.2. However, an analysis of the data analytics tool landscape suggests different categories for tools in the data analytics layer.

Agneeswaran identifies three generations of advanced analytics tools in [Agn14] and labels them 1st, 2nd, and 3rd generation [Agn14, pp. 10ff.]. However, the ability of advanced analytics tools to scale out, i. e., to distribute analytics workload, serves as a more specific discriminator. Moreover, advanced analytics tools cover contemporary predictive, but also descriptive analytics methods. Enabling predictive analytics often necessitates building a prediction model using, e. g., machine learning. Several tools support a standardized specification and exchange format called *Predictive Model Markup Language (PMML)* [154]. As aggregate and descriptive measures are handled in the data processing layer, OLAP functions remain to be covered. These functions can be categorized as traditional BI analytics, as typical OLAP functions use descriptive measures on multi-dimensional models. The summary of selected tools for the tool repository is listed in Table 4.4.

OLAP Tools provide OLAP functions as defined in the functional reference architecture. Although basic OLAP operators may be supported by recent SQL standards (cf. Section 2.1.1), dedicated OLAP tools offer support for many facets of OLAP not covered by SQL alone, such as pre-aggregation and sophisticated data cube designs. Microsoft SQL Server Analysis Services (SSAS) [271] and Mondrian [206] are selected to represent dedicated OLAP tools. Notably, Mondrian is also used in other BI suites such as the Pentaho BI Server[2].

[2] https://sourceforge.net/projects/pentaho/.

Centralized Analytics Tools denote "traditional" advanced analytics suites and tools such as IBM SPSS[3], RapidMiner [340], Weka[4] [242], or the R language with various scientific packages. Several of these tools also feature a graphical interface, which makes them suitable for end users with limited programming skills. Notably, several of these, such as IBM, RapidMiner, or KNIME are identified as leaders in the analytics area by Gartner (cf. [325]). Centralized tools are focused on non-distributed analytics, where the workload remains on a single machine conducting the analysis. Here, the limiting factors are the speed and capacities of main memory, disk, and processor cores.

Distributed Analytics Tools can distribute their analytic workload to clusters of computer nodes and are horizontally scalable. According to Agneeswaran, the main differentiator between the 2nd and 3rd generation of distributed advanced analytics tools is if analytics is conducted directly on Hadoop MapReduce or on more recent processing frameworks. In 2014, Agneeswaran found gaps in the analytic capabilities of Apache Mahout and RapidMiner Radoop [341] running on MapReduce exclusively [Agn14, pp. 10f.]. The 3rd generation of "beyond Hadoop" analytic tools utilizes engines such as Apache Spark with MLib or Storm and offered more algorithms and analytics methods than, e. g., Apache Mahout. Moreover, an execution of complex algorithms could result in many MapReduce jobs, which would have rendered computation inefficient because management time overheads. In contrast to MapReduce, newer frameworks also support real-time analytics and efficient iterative algorithms [Agn14, p. 11] [LKRH15, p. 11]. For example, Spark, Flink, and H_2O [195] — which is also considered in the Gartner hype cycle for data science platforms [325] — all conduct analyses in-memory, while MapReduce relies on regular disk storage [LKRH15, p. 10] (cf. Section 2.2.6). However, Apache Mahout has redesigned its execution engine to run Spark, Flink, and H_2O since that and marked its MapReduce implementation as "deprecated". The second notable example, RapidMiner

[3] https://www.ibm.com/analytics/data-science/predictive-analytics/spss-statistical-software.

[4] Weka 3.8 or newer allows to conduct data mining on Hadoop MapReduce or Spark, albeit with some limitations [389].

Table 4.4: GOBIA tool repository for the Data Analytics layer.

Technology Class	Tool
Centralized Analytics Tools	Anaconda (R & Python)[xviii] RapidMiner (Standalone) KNIME Weka (Standalone)
Distributed Analytics Tools	RapidMiner (Radoop)[xix] Apache Mahout Spark MLib H_2O (YARN / Spark)[xx] FlinkML Apache SAMOA Apache MADlib
OLAP Tools	Pentaho Mondrian Microsoft SSAS

[xviii] Enterprise distribution for R and Python. [xix] Official RapidMiner extension. [xx] Together with Spark known as *Sparkling Water*.

Radoop, an extension for standalone RapidMiner, also added support to Spark as execution engine and promotes its prominently [341]. Thus, a distinction into 2nd and 3rd is longer necessary for this model. This leaves *Distributed Analytics Tools* as second technology class for advanced analytics in the analytics layer.

4.3 GOBIA.DEV

The GOBIA.DEV development process aids in transforming initial BI goals and requirements into a set of feasible customized BI architecture alternatives. The goals and requirements are determined outside the GOBIA.DEV process in the organizational context around it (Section 4.1.2). A customized BI architecture is a blueprint for a detailed evaluation and implementation

process that follows after GOBIA.DEV has concluded. GOBIA.DEV ensures that the output is aligned to the input goal by using an iterative approach that can refine the input goals if necessary.

GOBIA.DEV is divided into three phases. Each phase helps to shape a more specific BI architecture. The three phases result from the analysis of analytical architectures in Chapter 3. Derived from the architecture level of detail naming template by Angelov [AGG12], the outputs are an *abstract BI architecture* using the functional reference architecture from GOBIA.-REF, which is further refined in the second phase into a *semi-concrete BI architecture*. Lastly, the third phase selects suitable technology sets, which can implement the semi-concrete BI architecture. This yields a set of alternative concrete BI architectures, termed *customized BI architectures*. This process is illustrated conceptually in Figure 4.4.

Each of these phases is elaborated using a similar structure. First, the overall phase goal is outlined which summarizes the intent of the phase with respect to the overall GOBIA.DEV process. Second, input artifacts and output artifacts are described so that the format of these is transparent to a potential user. Just as proper documentation of programming APIs provides transparency about parameters and output and enables a programmer to properly use a function, the description of the in- and outgoing interfaces of the phases has the goal to provide the same form of transparency here. Third, the process steps inside the phase are described and argued for. Particularly, any decisions that are made, respectively should be made while going through the respective phase are outlined. Finally, the execution of each phase is immediately illustrated using the FROG AIR case as running example.

4.3.1 Phase I

Phase Goal and Overview. The goal of this first phase is to create an abstract, functional representation of the target BI architecture, which is comprised of BI functionality and data entities as defined in the GOBIA.-REF functional reference architecture. This means that the elements of the aforementioned reference architecture are directly integrated into GOBIA.-DEV. The elements are selected according to BI requirements and goals and

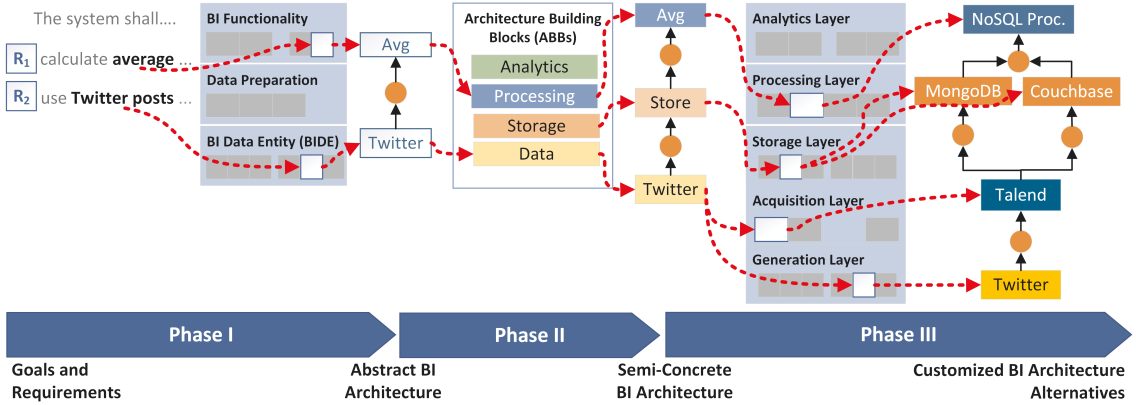

Figure 4.4: Conceptual illustration of the GOBIA.DEV process execution. An example involving calculating average values from Twitter data is used to illustrate the development starting with goals and requirements. Choices through GOBIA.DEV are depicted with dashed arrows. At first, an abstract BI architecture is created using the GOBIA.REF functional reference architecture in GOBIA.DEV Phase I. Subsequently, the next two phases further substantiate the abstract BI architecture until alternatives for concrete and customized BI architectures are formed. These alternatives are used after the GOBIA method for evaluation and selection.

should fulfill them and represent the proper BI-related "raw ingredients", i. e., artifacts that constitute a solution to the specific input BI requirements and goals of the case at hand. BI requirements and goals are the requirements and goals which refer to the to-be-implemented BI architecture, respectively system. The basic process flow of Phase I is depicted in Figure 4.5. An illustration of the process using the FROG AIR case can be found at the end of this section.

Both BI functionality and BI data entity are understood as defined in the GOBIA.REF functional reference architecture, namely as conceptual elements to abstract from any specific technology, so that the focus can be on the needed functionality and data entities. Thereby, it is also specified how functionality and data are connected to one another, e. g., which data entities are generally needed for the BI functionality to work. In the process of figuring out these two, a third type of element can also be specified: a data preparation step. It poses a high-level conceptualization of tasks that transform the data into the format needed for the BI functionality. In summary, the data preparation steps ensure that BI functionality and BI data entity are properly aligned to one another, i. e., that they fit to each other. Furthermore, they provide a basis to specific more detailed processing tasks later on.

Inputs. Both *initial goals and initial BI requirements* are the inputs that start GOBIA.DEV Phase 1. They are captioned *initial*, because they can be further refined or changed in the GOBIA.DEV process if need be (e. g., if a requirement cannot be satisfied or needed data is economically unfeasible to be acquire). GOBIA.DEV is not limited to a specific RE methodology as predecessor activity (e. g., the ones outlined by [Poh10]). Instead, GOBIA.-DEV should be compatible any individual form of acquiring initial goals and BI requirements. Therefore, the aforementioned do not need to be fully specified in every aspect possible prior (e. g., as done in traditional approaches such as the waterfall or V-model). However, they need to adhere to certain minimum requirements regarding their form and content. At first, a name or description are sufficient as long as the provided BI functionalities and BIDEs can be selected. Notably, it is assumed that *constraints* referring to functional

and non-functional requirements are attached to these. For example, when a token in the process model contains an object that describes a respective requirement, a constraint is added as metadata to this object. Alongside that, other meta properties can be added, e. g., relationships to other requirements and goals. Notably, the latter are optional information at this point.

Optimally, these goals and requirements are exhaustive, i. e., no essential goals or requirements are generally missing, given the knowledge about the case at this point in time. Usually, this is ascertained thorough a preceding requirements engineering activity involving the relevant stakeholders, where first goals, and then the requirements based non these goals, are commonly agreed upon. The responsibility to ensure that these goals and requirements are properly defined and documented (e. g., that they are free from contradictions and atomic) also falls into the realm of requirements engineering (cf. Section 2.5). While executing GOBIA.DEV, several "stops" are implemented, where the created elements can be checked if they can fulfill the requirements and goals. GOBIA.DEV also explicitly allows to amend and alter the goals and requirements if these cannot be fulfilled with the created artifacts. Ultimately, the decision whether formulated goals and requirements are commonly exhaustive in a use case at a given point in time, cannot be made automatically and needs to be made by humans with proper expertise. Such experts can determine if a textual or visual model, created using goals and requirements, is an adequate representation for a given use case.

Apart from that, if users of GOBIA are unable to complete a phase, e. g., because information is missing for an activity, the process can only continue if this information is acquired. GOBIA.DEV execution needs to be paused or aborted if a lack of information prohibits further progression. Notably, the GOBIA.DEV descriptions here state which information is required and what is decided upon in a phase so that it becomes clearer which level of detail regarding the use case is required. This can allow to leave out, e. g., technical details to a certain extent in the initial phases, where it is only needed in limited quantity. For instance, to discuss BI functionality, it is not yet necessary to know which form of delivery guarantees (cf. Section 2.2.6) in an SPT is adequate for the case.

Output. The main output of Phase I is an *Abstract BI Architecture*, which is a result of the customization of the GOBIA.REF functional reference architecture. This abstract BI architecture structures the three element types in three layers, which correspond to the element types, i. e., BI functionalities, optional data preparation steps, and BI data entities. The architectures are visually depicted as simplified Petri nets for additional consistency.

Process Steps

Extract Functional and Non-Functional Requirements. The first step is responsible for extracting the necessary functional requirements, which are the foundation to select both functionality and data entities in the subsequent step. In particular, non-functional requirements are side-lined as they are crucial to assess intermediary results using the whole set of requirements. If not already done during requirements elicitation, it is now necessary to specify to which category the requirements belong. Once this is done, the process can continue.

After all functional requirements are selected, only unrelated requirements remain. Instead of prohibiting the process from continuing, they are set aside and can be employed to align the intermediary elements later. *Unrelated* functional requirements do of course refer to the BI system, but they are unrelated in a sense that they do not directly impact selection functionality, preparation, or data specifically.

Definition of BI functionality. In this activity, a functional requirement is mapped to appropriate predictive, descriptive, or non-analytic BI functionality as defined GOBIA.REF functional reference architecture (cf. Figure B.1). A functional BI requirement may be mapped to one or to several activities. Although a definition of requirements using dedicated methods as outlined in Section 2.5 should result in requirements, which do not subsume several distinct functionalities, complex analytics may consist of several steps. For example, a classification could be predecessor activity for association rule mining (cf. Section 2.3.3). To conduct the mapping, the name and description of a requirement need to be studied. First, it must be decided if a value or

category should be predicted. Otherwise, a descriptive analytics method or a simple relaying of information as non-analytic functionality is fitting. Requirements need to be mostly broken down to the most essential analytical functionality. This might reveal missing details in the requirement specification. For example, a requirement that a system should "recommend" a certain item to a customer, must clarify which analytical functionality is meant by "recommend". While collaborative filtering is a typical method in this regard, the actual intention could also be to recommend the most profitable item, which indicates the calculation of a maximum value inside data. This is covered by the functionality block *Aggregate Measures & Descriptions*. Table 4.5 lists example expressions, which could aid in mapping a requirement to BI functionality (i. e., possible verbs and other activities mentioned inside requirement specifications). Currently, this mapping is conducted by users executing GOBIA.DEV. Future options to automate this mapping such as NLP are discussed separately (cf. Section 7.2). Additionally, constraints to selecting functionalities have to be considered. These are annotated besides the functional requirements and might restrict the solution space, which is represented by the list of BI functionality. The same approach is applied in the next step.

Definition of BI data entities. To define BIDEs as specified in the GOBIA.-REF functional reference architecture (cf. Section 4.2), information of data used for the respective requirements is extracted from its specification, where applicable. A physical data source does not need to be explicated in this phase. It is sufficient, when high-level conceptual sources are mentioned. For instance, instead of needing to specify a specific Twitter API or source system, a simple *Twitter data* BIDE can be created. Already known information regarding the physical sources might be noted alongside the created entity as metadata. What is decided at this point are data variety and data velocity. Regarding the latter, it should be stated if a data entity is resting data or streaming data. This is independent from latency requirements of the BI system. This aspect is decided by the properties of the sources. A data entity is a resting BIDE if the sources provides data infrequently or data is accessed rarely. The time spans for data latency depicted in Table 2.5

Table 4.5: Sample expressions and words, which may be found in requirements descriptions and help to indicate which BI functionality should be defined for the respective requirement. The expression consists either of synonymously used words, e. g., *categorize* for classification, or of specific sub-activities, e. g., *average* for aggregate measures or *roll-up* for OLAP functions.

BI functionality	Sample expressions and words in requirement descriptions
Classification	Categorize, Classify, Attach label, Predict category, Predict class
Regression	Estimate, Predict value, Forecast value, Perform regression
Time-series analysis	Analyze time-series, Trend analysis
Collaborative filtering	Collaborative filtering, Recommendations based on items, ... based on users
Aggregate Measures & Descriptions	Count, Average, Sum, Minimum, Maximum, Variance, Deviation
OLAP Functions	Drill-down, Roll-up, Pivot, Slice, Dice, Multi-dimensional analysis, OLAP, Reporting

can be used to guide this classification. Regarding variety, a disambiguation into unstructured, semi-structured, and structured data is necessary. This can be done, e. g., by consulting Table 2.3 or collecting input from experts and stakeholders. The resulting data entity can be named freely, while the element type is added as annotation to the entity (e. g., *Twitter data* is of type *Semi-structured BIDE*).

Check alignment between BI functionality and BIDE. This step aims to determine whether functionalities and data for a requirement are consistent with respect to each other. This internal consistency is given, if no high-level data preparation steps are needed to use the defined data in the functionalities it is used for. The conceptual data preparation steps can be dimensional reduction, filtering of data points, a combination of sources, or

normalization of values (e. g., units of measurement). Dimensional reduction should be selected if the source exhibits high dimensionality, which is not necessary or prohibitive for analysis. Attributes not needed can be left out. Filtering activities remove data points (e. g., rows) from the data set. If only a known subset of information is deemed necessary (e. g., Twitter posts made by own customer representatives), a filter step can be chosen. Normalization, for instance, unifies value scales such as centimeters and inches. A combination of sources can be placed when several data entities should be joined or otherwise combined for joint usage.

Check elements alignment to goals and requirements. Once internal consistency is ensured, it is checked whether the created elements are *generally* able to satisfy the goals and requirements or whether there are either elements missing or goals and requirements are underspecified. Naturally, a complete assessment according to quality or acceptance criteria is not possible at this point. Rather, discrepancies that generally hinder goal and requirement fulfillment should be detected. BI functionalities or data entities are missing if there exist relevant requirements, which are not covered by the created elements. In any case, it is assumed that issues with requirements and goals contribute to such discrepancies. To determine if goals or requirements need refinement, the *Assess Mitigation* step is executed. It decides whether either the overall goals of the BI system needed to be further refined or whether the specification of the requirements has to be improved. In the former case, the Phase I process ends and a redefinition of goals at the organizational level needs to take place (cf. Figure 4.1). Then, if Phase I is re-executed with refined goals, they become an optional input to this activity. In the latter case, an activity to improve the BI requirements needs to be executed, which result in an updated set of BI requirements accompanied by the initial goals. This general approach to aligning elements and requirements to each other is called *co-alignment*. First, elements fitting the requirements are attempted to be defined. If this is not sufficient, the requirements or goals are altered to enable better specification of BI functionalities, data preparation steps, and BI data entity.

Table 4.6: Initial input data items from the FROG AIR case to start the GOBIA.DEV process.

Input	Artifacts from the FROG AIR case
Initial Goals	$G_1, G_{2.1-2.2}, G_{3.1-3.5}, G_4, G_5$
BI Requirements	R_1 – R_{15}

Compose Architecture. At this stage, a set of BI functionalities, data preparation steps, and BI data entities has been created, which are both internally consistent and externally aligned to their goals and requirements. Using these elements, the *Abstract BI Architecture* is composed as outlined above.

Application of Phase I to the FROG AIR Example

The goals and requirements for the FROG AIR case (cf. Section 1.3) are the initial input for Phase I. In total, there are 10 goals and 15 requirements (cf. Table 4.6). Here, these goals and requirements were agreed upon beforehand, and thus they are assumed to be exhaustive at this point.

In the following, the execution of Phase I of GOBIA.DEV for FROG AIR is demonstrated. This is done by describing each process activity and decision with respect to the FROG AIR case. This documentation is illustrated by process visualizations that consider, e. g., the specific goals and requirements that are processed by Phase I. This approach is later repeated for the demonstration of Phase II and Phase III.

Extract Functional and Non-Functional Requirements. As the distinction into these requirement types has been made before for the FROG AIR case, the result of this step is easily deductible (cf. Figure 4.6).

Define BI functionality and BI data entity. For each functional requirement, a fitting BI functionality or BI data entity must be selected from the underlying GOBIA.REF model. R_{13}, which is concerned with visualization, is neither BI functionality nor data entity and is put aside for the following activity. R_5 relates to a needed processing functionality, which is not dealt

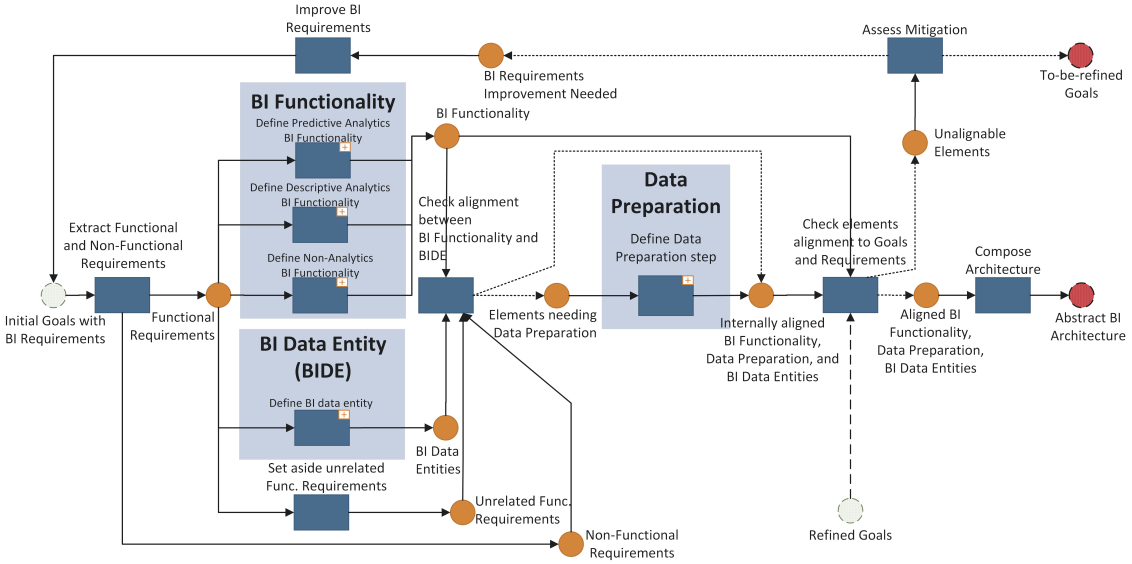

Figure 4.5: Petri net process model of GOBIA.DEV Phase I. Details of BI Functionality, Data Preparation, and BIDE blocks are abridged. The full process is shown in Appendix B.

Table 4.7: Separation of input data into GOBIA.DEV Phase I into functional and non-functional requirements as first step.

Requirement Type	Requirements from the FROG AIR case
Functional Requirements	$R_1 - R_{14}$
Non-Functional Requirements	R_{15}

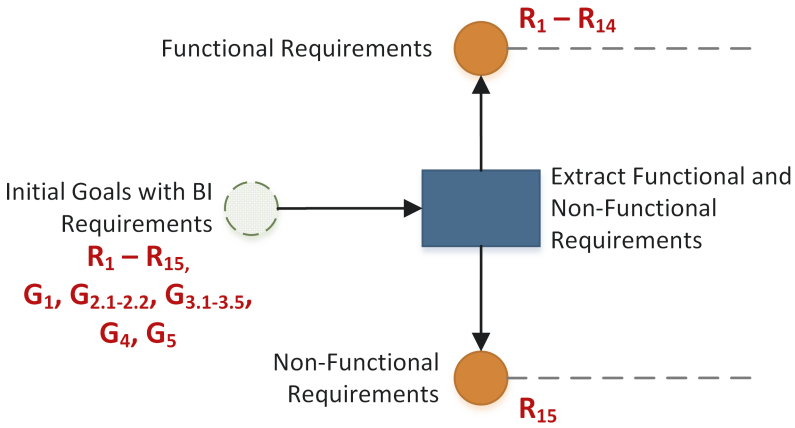

Figure 4.6: First part of the execution of GOBIA.DEV Phase I in the FROG AIR case.

with at this point as well. R_{14} is set aside as it is unrelated to BI functionality or BI data entities. The same applies for R_1 and R_2, which are front-end settings for the user. R_3 and R_4 relate to data, where as the remaining R_{6-12} concern BI functionality. Regarding data, each requirement refers to a distinct data set.

Conceptually, the BI data entities *Twitter messages* (R_3) and *Facebook messages* (R_4) are created, as this data is needed to satisfy the respective requirements. The underlying physical data sources can be mapped one-to-one to these entities in the FROG AIR case, as all data is accessed remotely (e. g., via a REST API call). This representation should be similar for most publicly available social platforms, e. g. VKontakte in Russia, or other enterprise-enabled messengers platforms, e. g. WhatsApp as they need to be regarded externally accessible data storage.

To map the requirements to BI functionality, the appropriate BI method is deducted from title and description of the requirement. Here, the BI functionality can be deduced from their respective names. For instance, R_{10} to R_{12} all contain a reference to "counting" in their names. Thus, the corresponding BI functionality selection activity *Aggregate Measures & Descriptions* is activated, because this encompasses counting elements as aggregate measure (cf. Table 4.5). Notably, it is done only once for all three activities as they can be grouped together as they relate to the same data to be counted. For R_9, the requirement name mentions "average response time" which also corresponds to an average aggregate measure in the process. The labels for the BI functionalities are chosen to relate to the requirements they are based upon. The process is illustrated in Figure 4.7.

Check alignment between BI functionality and BIDE. There is no connecting requirement between, e. g., R_9 and R_3 or R_4 necessitating a data preparation step. Indeed, inspecting exemplary data from the actual "raw" messages from Twitter and Facebook reveals that not all posts, e. g., on the Facebook wall of a company account, constitute actual questions. Rather, these can be positive proclamations (e.g., thanking for a successful flight). Thus, the actual relevant messages need to be filtered. A "Filter" preparation step is added, which filters out irrelevant messages, which did not require

Table 4.8: BI functionalities and BIDEs for the FROG AIR case derived in GOBIA.DEV Phase I.

Element Type	Identifier	Name	Element
BI functionality	BF_1	**Sentiment Analysis** of customer messages	Sentiment Analysis
	BF_2	Calculate **average** response time	Aggregate Measures & Descriptions
	BF_3	**Count** customer interactions	Aggregate Measures & Descriptions
BI data entity	$BIDE_1$	Twitter messages	*Semi-structured* BIDE
	$BIDE_2$	Facebook messages	*Semi-structured* BIDE

Table 4.9: New requirements R_{16} and R_{17} for the FROG AIR case added in GOBIA.DEV Phase I.

Requirement	Description	Type	Requires
R_{16}	*Filter irrelevant Facebook messages.* The system shall filter out Facebook messages that do not stem from interactions with customer service agents prior analysis.	Functional	R_5
R_{17}	*Filter irrelevant Twitter messages.* The system shall filter out Twitter messages that do not stem from interactions with customer service agents prior analysis.	Functional	R_5

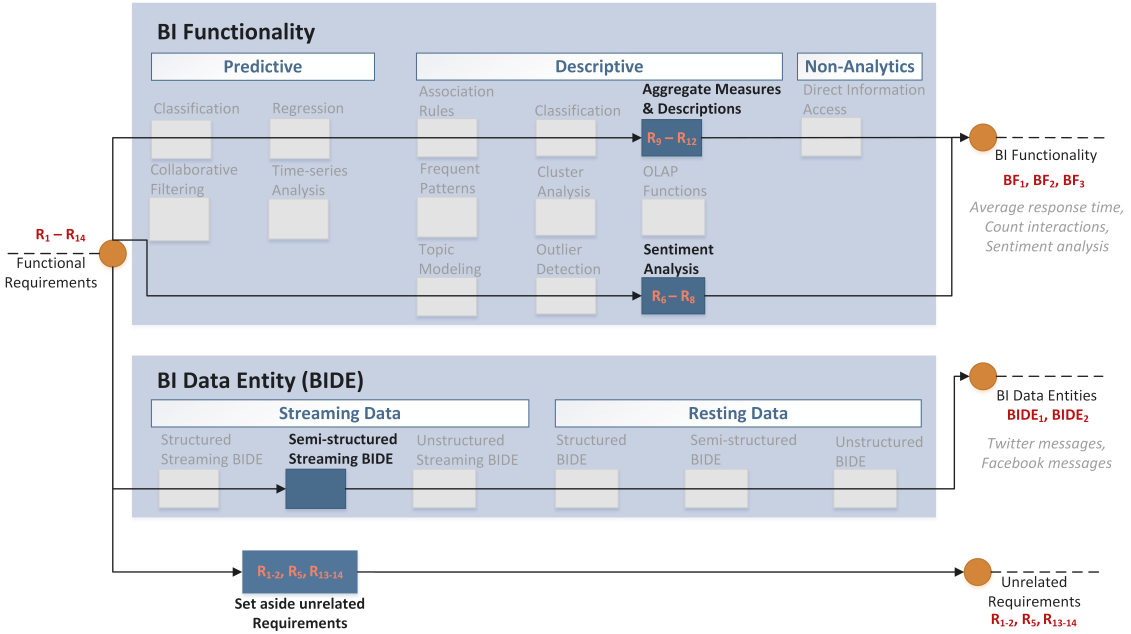

Figure 4.7: Second part of execution of GOBIA.DEV Phase I in the FROG AIR case: Selection of BI functionality and BI data entities.

Table 4.10: Data preparation steps added for the FROG AIR case in the course of GOBIA.DEV Phase I.

Element type	Identifier	Name	Inputs	Outputs
Data preparation	DP_1	Filter irrelevant Twitter messages	$BIDE_1$	DP_3
	DP_2	Filter irrelevant Facebook messages	$BIDE_2$	DP_3
	DP_3	Join messages, retaining origin information	DP_1 or DP_2	BF_1, BF_2, or BF_3

an answer by a service agent. In this case, it also purposeful to amend new requirements R_{16} and R_{17}, which deal with this preparation task directly (cf. Table 4.9). New requirements are purposeful, because adding this aspect to the existing requirement R_5 would violate the recommendation that requirements should be atomic (cf. Section 2.5).

Check elements alignment to requirements. Next, consistency between the created elements (BI functionality, data preparation, BI data entity) to the requirements set aside before and to the non-functional requirements is checked. The specified non-functional requirement does not necessitate further alignment. This respective performance requirement R_{15} does not form a consistency hazard at this stage, because it does not concern the basic functionality or data elements in the conceptual phase. This requirement becomes more relevant in the course of the next phases and in the following implementation after the GOBIA method has concluded. This applies to functional requirement R_{14} as the necessity to have web front-end is not evaluated in GOBIA.DEV. Likewise, the set-aside requirements R_1, R_2, and R_5 also do not impose constraints or quality criteria to be considered at this point.

Compose Architecture. Therefore, the final abstract BI architecture can be composed with elements created before by grouping the elements together

that are connected respectively dependent on another. In this case, all BI data entities are connected to BI functionality via data preparation steps. However, this is not a necessity in general. The result is depicted in Figure 4.9.

4.3.2 Phase II

Phase Goal and Overview. The second of phase of GOBIA.DEV aims to refine the abstract BI architecture from first phase into a semi-concrete BI architecture. A semi-concrete BI architecture adds technical details and refines previously defined elements by mapping them to more concrete so-called ABBs. ABBs are on a more technical layer below the high-level functional architecture and extend it with aspects, which are more closely focused on a technical view on the BI functionalities to be realized. These activities ingrain the desired architectural patterns for the target BI architecture into the abstract BI architecture and illustrate the flow of information between functionality, data, and persistence. Patterns highlighted in Section 3.5 can be readily applied here. In addition to that, customized forms of the aforementioned or fully customized patterns can be used to define the semi-concrete BI architecture. In particular, decisions about data persistence (i. e. about storing data or not) are made here. For each BI functionality, its chain of subordinate activities is refined. While doing this, the three element types from the abstract architecture are transformed into one or more ABBs. These building blocks can be used to directly assign a specific piece of technology to them in the following third phase. The Architecture Building Blocks in this phase are *analytic*, *processing*, *storage*, and *data generation* ABBs. They are chosen according to value chain activities pictured in Section 3.2.2 and typical layers found in other reference architectures (cf. Chapter 3). Naturally, BI functionality and data preparation steps can be directly mapped to analytic, respectively processing building blocks. However, it is necessary to consider possible technical realizations, which may lead to additional building blocks to be added. For instance, a processing building block may be necessary to transform data from a previous building block into a format that can be used by a subsequent analytic building block. Also, depending on the BI data entities and how they are generated and acquired, storage building

blocks may be required to persist data at various points. For example, for streaming data with strict latency requirements, a storage building block is likely to be counter-productive. Notably, even if several storage or other building blocks are used, the same technological artifact may be used for their overall realization later on. Several storage building blocks do not imply that a different storage solution must be used for each of them, but rather that storage is required at various points in the process. This opens up the opportunity to select the best pieces or piece of technology based on more specific evaluation criteria later on.

Another notable difference between the abstract BI architecture and the semi-concrete BI architecture is that the former follows a strict layered approach with respect to the GOBIA.REF functional reference architecture, while the latter can alternate between various types of building blocks as needed to realize the target BI functionality. This in accordance to identified data flows in a unified Big Data value chain (cf. Section 3.2.2) as well as the result of the analysis of architectural patterns. These patterns exhibit a more dynamic nature than traditional Data Warehouse patterns. The overall process is visualized in Figure 4.8.

Input and Output. Primary input is the abstract BI architecture created in Phase I. In addition to that refined goals can be injected into this phase as optional input, which can further guide this phase. Refined goals are created in this phase, when the resulting building blocks cannot be composed into an architecture and a refinement of goals is needed. Output of Phase II is a semi-concrete BI architecture.

Process Steps

Separate into BI functionality, data preparation, and BI data entity elements. This step, which separates the three elements types from the output of the previous phase, aims to accomplish two goals. First, the abstract BI architecture is separated by BI functionality so that steps to achieve each functionality are made transparent. In the end, each BI functionality should is responsible for part of value generation, while the remaining

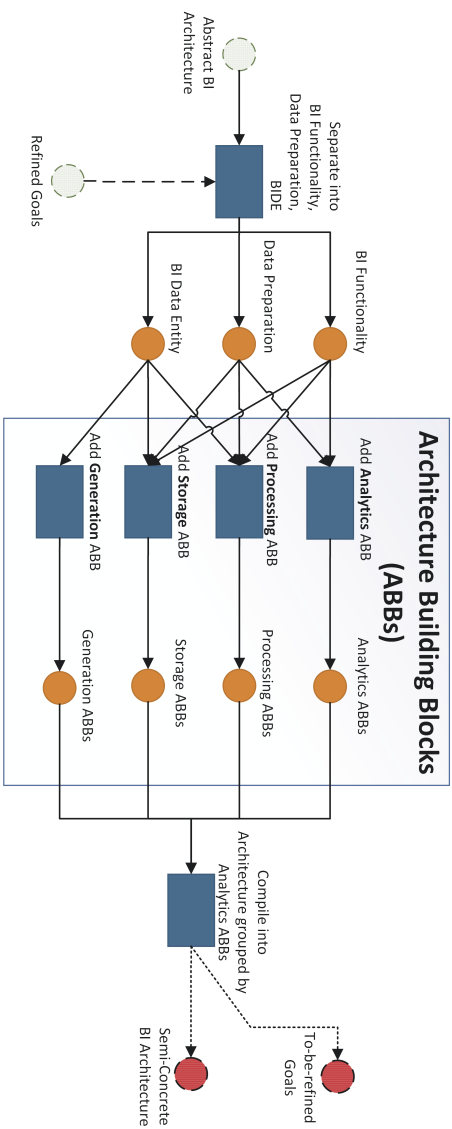

Figure 4.8: Petri net process model of GOBIA.DEV Phase II.

elements needed to realize this BI functionality can be seen as sequence of steps along the architecture layers leading up to it (as depicted in a value chain, cf. Section 3.2.2). Second, this separation ensures that each element of the abstract BI architecture can be transformed into a building block individually. Additionally, the elements are explicitly separated by their type (BI functionality, data preparation, and BI data entity).

The value generation leading up to a specific BI functionality as chain of activities along the layers of an architecture is termed *tactical plan*. The term is derived from the notion that a strategic plan resembles the most abstract form of a plan, similar to the abstract BI architecture in Phase 1, which is the most abstract BI architecture in GOBIA.DEV. Moreover, the equivalent of a more detailed operational plan are the concrete BI architecture alternatives, which result from Phase III. A tactical plan groups activities by final BI functionality. This is the functionality which ultimately represents a specific usage scenario, e. g., a sentiment analysis. This separates value chain activities for each ultimately realized BI functionality. Together these tactical plans are ultimately organized into a semi-concrete BI architecture model. However, technology selection is driven by each of these plans, which implement a data value chain. Notably, this tactical plan is represented through the inherent links between individual BI functionality, data preparation steps, and BIDEs tokens in the process model.

Add Analytic/Processing/Storage/Generation ABBs. For each element from the abstract BI architecture, ABBs are generated. *(Data) Generation* building blocks represents data sources, which are fed into the system. In contrast to conceptual BIDEs, actual places of data generation can be created here. The naming of these building blocks is derived from the unified Big Data value chain (cf. Section 3.2.2). Besides that, no additional information is needed, as data velocity and variety have already been determined for the BIDEs in Phase I. The building blocks also carry properties of the source data, which is needed to paint a complete picture of the architecture and improve technology selection. *Analytic* and *Processing* building blocks represent corresponding value chain activities. The same applies to storage *Storage* building blocks. However, this block can be placed wherever data

is persistence is needed or desired, i. e., during the mapping of each block. This resembles the role of storage as optional, but sometimes necessary support functionality (cf. Section 3.2.2). If functionality is not atomic, it can be broken down into several ABBs to separate concerns. It should be noted that the same technology can later be selected for several consecutive ABBs. If the same type of ABB is used in a simple sequence, technology decisions do not need to be repeated for each entry, when it can be assumed that the functionality belongs to the same pipeline of functions. For example, several subsequent transformations on streaming data can lead back to the technology choice of the first ABB, if appropriate. In particular, non-analytic BI functionality (cf. Section 4.2.1) can be interpreted as extension of the previous data processing or analysis pipeline, which integrates and prepares the data.

Mapping of BI functionalities. For BI functionalities, it has to be decided if their basic functionality is technically an analytics or a processing functionality. Analytics ABBs are to be used, when either descriptive or predictive analytics are to be used. While most of these BI functionalities can be mapped to a corresponding analytics building block, especially *aggregate measures & descriptions* are functionalities typical covered by simple data processing, e. g., by SQL. Such elements should be mapped to processing ABBs instead. Additional building blocks can be generated if further technical processing is required.

Mapping of data preperation steps. Often, steps to prepare data can be mapped to processing ABBs. For example, *filtering* rows, *combining* or joining data, or *removing attributes* can be expressed as data processing activity, which are algorithmically simple transformations on data. These are covered by typical data processing systems such as RDBMS with SQL or SPEs. Complex dimensional reduction techniques necessitate analytics methods (cf. Section 2.3.3), which cannot be implemented by simple means and result in an analytics ABB.

Mapping of BI data entities. Conceptual entities are mapped to generation blocks as outlined above and storage ABBs can be added. For

example, if data should be immediately stored as-is in a data lake or similar structure, a storage block can be added directly after a generation block. If no immediate storage is needed, only a generation ABB is created. Apart from that, data combinations of several data generation blocks might necessitate processing activities. For example, BIDEs that are generated based on external sources can be covered through processing and storage ABBs after a generation ABB for the actual source data.

Each created ABB should have a unique identifier such as *(1.5)*[5], a type identifier such as *Processing* and a unique name referring to the content of the ABB, e. g., *Store aggregate measures*.

Compile into Architecture grouped by Analytics ABBs. After all individual elements from the abstract BI architecture have been transformed into building block sequences, they are composed into distinct tactical plans grouped by their primary BI functionality. This is a technical activity as the previous step already implicitly regarded building blocks not only in isolation, but also their subsequent and preceding building blocks. Using these, the semi-concrete BI architecture with all tactical plans can be created, which structures ABBs into distinct layers. Before the final semi-concrete architecture is passed on to the next phase, it is **checked for alignment** against the input goals and requirements. Considering the data flow with added persistence, it has to be verified whether the specific tactical plans the architecture result in a conflict with the goals. For example, if real-time responses are needed, but several storage blocks are added, they might hinder the fulfillment of this goals. Similarly, when persistence is required, but no storage is added, the achievement of goals is hindered — or if discrepancies between actual data sources and BIDEs are not resolved, a refinement is needed. A refinement of goals ultimately triggers this or a previous phase or halts the process, if alignment cannot be ensured. Therefore, the result of this activity is either a semi-concrete BI architecture, which leads to the next

[5] In this case, this might be used to denote the fifth ABB for the first tactical plan, which is based on the first BI functionality.

phase, or a mis-alignment to goals and requirements, which necessitates a re-refinement and re-evaluation of said project goals.

Application of Phase II to the FROG AIR Example

The abstract BI architecture in Figure 4.9 depicts the needed BI functionality in a structured manner. In Phase II, this representation is refined and amended with additional details, especially regarding data storage. This transformation using ABBs is done in accordance with the output from the previous phase and in alignment with the current set of goals and corresponding requirements.

Separate by BI functionality and subordinate data preparation and BI data entity elements. Figure 4.9 visualizes the initial separation of elements by tactical plans as a value chain, whereas the same element types are grouped by color. Clearly, there are three distinct use cases (one for each BI functionality), which are to be created in this phase: Sentiment Analysis BF_1, Calculate average response time BF_2, and Count customer interactions BF_3. The corresponding tactical plans are named TP_1, TP_2, TP_3 accordingly.

Notably, each of these plans has a common denominator, as they rely on the same prepared data: Filtered (DP_1, DP_2) and joined (DP_3) messages across Twitter and Facebook. In the following, the data preparation part until Join messages (DP_3) is discussed at first, before elaborating upon the details of the three tactical plans (i. e., BI functionalities). This data preparation part shall also be referred to as *preparation chain* onwards.

Add Analytic/Processing/Storage/Generation ABBs. A sequence of building blocks is created for each target BI functionality. Inside each sequence, all conceptual plan elements preceding the target BI functionality are transformed into an adequate set of building blocks. Most elements of Figure 4.9 for the FROG AIR case can be directly transformed into one building block. However, additional storage building blocks are added on top of that. Building blocks all have a unique identification number (e. g., 1.1 or 2.5), which is derived from the tactical plan they belong to and their order in the overall

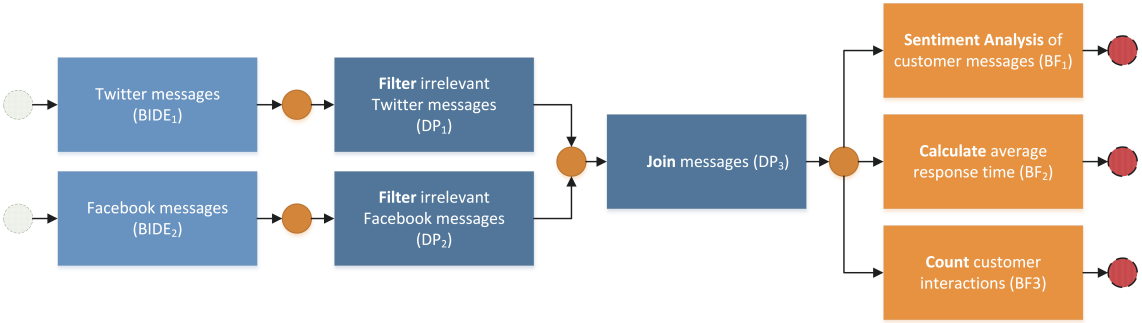

Figure 4.9: GOBIA.DEV for the FROG AIR case: Abstract BI architecture of Phase I depicted using a Petri net with the three element types BIDE, data preparation, and BI functionality ordered from left to right.

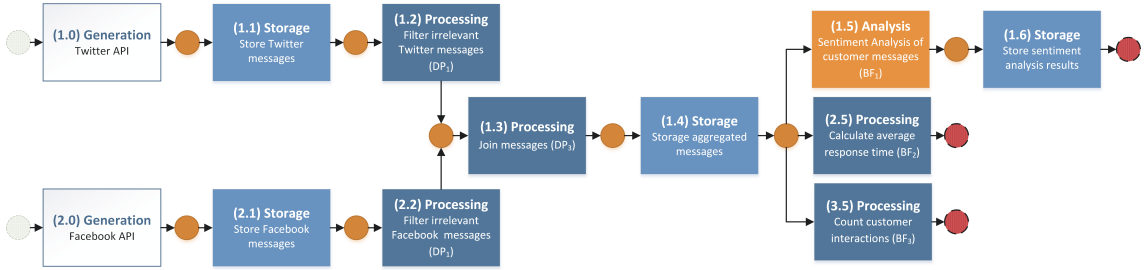

Figure 4.10: GOBIA.DEV Phase II for the FROG AIR case: Tactical plan perspective of the semi-concrete BI architecture, which is the main output of Phase II. A unified view on tactical plans outlines value generation using ABBs.

execution sequence. The *data preparation chain* has the same identifier for all tactical plans due to it being shared among them.

First, the common data preparation chain of the Twitter and Facebook messages is dealt with, because it is shared among for all three tactical plans, which are to be created.

Instead of BI data entities for Facebook, respectively Twitter ($BIDE_{1-2}$), the tactical plans are started with a generation building block, which represents Twitter and Facebook data retrieved via their respective APIs and a storage building block that is responsible for storing scraped messages first. With this, an external component can scrape and provide the data to the system. This has the advantage that the filtering and joining pipelines afterwards can work asynchronously from the data acquisition. Because there are only access time restrictions for the front-end (cf. R_{15}), a potential delay in capturing the latest data is deemed acceptable. Furthermore, considering availability restrictions for older Twitter posts (cf. [388]), it seems favorable to store tweets and retain them. Thus, two storage building blocks are created as start of the preparation chain (cf. Figure 4.10). However, these storage building blocks 1.1 and 2.1 are in no linear sequence, but organized in parallel to each of other, as the activities are independent. The following filter activities DP_1 and DP_2 can be readily transformed into one processing building block with the same name (1.2 and 2.2 respectively). These two parallel lines are merged by the following processing building blocks that integrates the Facebook and Twitter messages into a unified form (1.3). To have a persistent storage of the processed and integrated data for the subsequent analytics building blocks, a storage building block is added as the last step in the preparation chain (1.4).

With the common preparation chain complete, the BI functionalities can be transformed into building blocks. Upon inspecting the specifics of each BI functionality, only the sentiment analysis BF_3 is regarded as analytical building block 1.5. The other two BI functionalities use *count* and *average* calculations, which are categorized as processing capability. Thus, they yield a processing building block respectively (2.5 and 3.5). For the sentiment analysis (1.5 / BF_1) a subsequent storage block is added so that the sentiment

Table 4.11: GOBIA.DEV Phase II: Tactical plans for FROG AIR.

Identification	Target BI functionality
TP_1	Sentiment Analysis of customer messages (BF_1)
TP_2	Calculate average response time (BF_2)
TP_3	Count customer interactions (BF_3)

analysis results do not need to be recalculated each time the respective information is requested. This is because a sentiment analysis is regarded as more computationally intensive and the storage should ensure that the latency requirement R_{15} can be met more easily.

Compile into Architecture grouped by Analytics ABBs. Figure 4.10 displays a compilation of all three tactical plans. Due to the similarity of the tactical plans, only one tactical plan TP_1 is visualized separately as an example (cf. Figure 4.11). Besides the very last activities, TP_2 and TP_3 are identical to TP_1. The three resulting tactical plans are depicted textually in Table 4.11. Their depiction as full semi-concrete BI architecture can be found in Figure B.5, located in Appendix B.

4.3.3 Phase III

Phase Goal and Overview. The final phase of GOBIA.DEV is responsible to identify appropriate layers of the GOBIA.REF technological reference architecture (cf. Section 4.2) and select feasible technology classes such as RDBMSs from these layers according to the previous results as well as the case goals and requirements. As intermediary result, technology sets are created, which contain instances of tools or processing approaches that implement desired functionalities with the given data. These instances are chosen from the GOBIA tool repository (e. g., MySQL and PostgreSQL as alternatives for RDBMSs). Notably, the tool repository only emits tools, whose capabilities match the given input. For instance, if an analytic tool in the repository does not support sentiment analysis, but such BI functionality was chosen, that particular tool is not returned. After attaining several sets

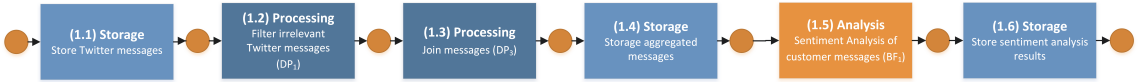

Figure 4.11: GOBIA.DEV Phase II for the FROG AIR case: Excerpt of Semi-concrete BI architecture for sentiment analysis BI functionality (TP_1).

Table 4.12: GOBIA.DEV Phase II: List of Architecture Building Blocks through all FROG AIR tactical plans.

ABB	Type	Description	Tactical plans
1.0	Generation	Twitter API	TP_{1-3}
2.0	Generation	Facebook API	TP_{1-3}
1.1	Storage	Store Twitter messages	TP_{1-3}
2.1	Storage	Store Facebook messages	TP_{1-3}
1.2	Processing	Filter irrelevant Twitter messages (DP_1)	TP_{1-3}
2.2	Processing	Filter irrelevant Facebook messages (DP_2)	TP_{1-3}
1.3	Processing	Join messages (DP_3)	TP_{1-3}
1.4	Storage	Store aggregated messages	TP_{1-3}
1.5	Analysis	Sentiment Analysis of customer messages (BF_1)	TP_1
1.6	Storage	Store sentiment analysis results	TP_1
2.5	Processing	Calculate average response time (BF_2)	TP_2
3.5	Processing	Count customer interactions (BF_3)	TP_3

of technologies, which represent various viable alternatives, compatibility between these choices is ensured using a provided comparability matrix of the tool repository.

Employing the information gathered in the previous phases, the ABBs from the semi-concrete BI architecture are mapped to the GOBIA.REF technological reference architecture model. To realize this mapping, each building block is mapped to one or two layers of the aforementioned model. While the assignment possibilities of building blocks to potential layers might seem limited, criteria are used to further determine which specific layers and specific technology classes will be considered for selection afterwards. These criteria originate from the use case at hand and are embodied in the goals and requirements (e. g., latency requirements, cf. Section 2.2.6) and

in the artifacts created until this point (e. g., a streaming data BIDE). The selection of criteria is based on the criteria synthesis in Section 3.6, which are derived from architecture analyses in Chapter 3 and generic properties of technologies elaborated upon in Chapter 2. For instance, to decide what kind of data acquisition technology is needed, the information if incoming data is streaming data or resting data will be utilized. The overall process is depicted in a simplified manner in Figure 4.12. The full processed is depicted in Appendix B.

Inputs and Outputs. Primary *input* is the semi-concrete BI architecture from Phase II. It encodes use case specifics in ABBs, which represent BI functionality and BIDEs. In this phase, ABBs are separated to find suitable choices for each of them. The final *output* of Phase III are customized BI alternatives. A final choice between these concrete BI architectures is made by means of a detailed evaluation under consideration of TORE factors after GOBIA.DEV has concluded. This is detailed in Section 4.4.

Process Steps

Decompose into Architecture Building Block. First, the semi-concrete BI architecture is separated into the four different ABB types. The reasoning behind this is that each ABB leads to the selection of one or two layers of embedded GOBIA.REF technological reference architecture. For each of these layers, an appropriate technology class is select. For each of these classes, technology instances are provided from the GOBIA tool repository. That way, a customized technology can be selected according to the needs of each building block. Notably, this selection also may factor in factors outside a specific ABB, where necessary.

Assign Layers. Now, the separated building blocks are mapped to respective technology layers from the GOBIA.REF technological reference architecture. Analytics, processing, and generation building blocks are mapped to their similarly named technology layers. However, data generation ABBs can be mapped additionally to an acquisition layers for technology selection. This resembles a further refinement of ingestion capabilities, where externally

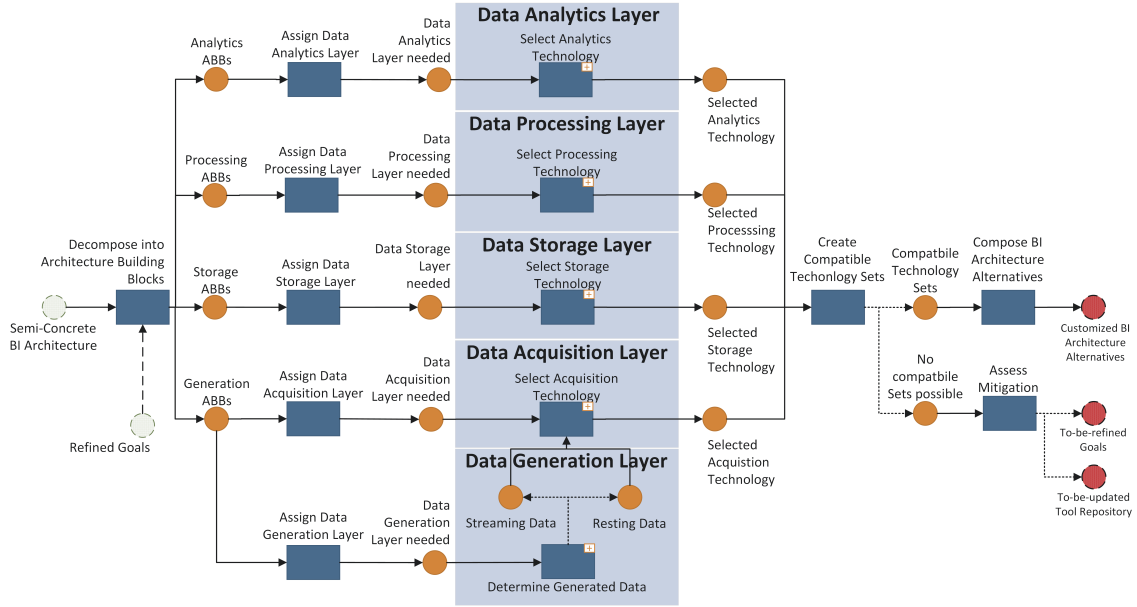

Figure 4.12: Simplified Petri net process model of GOBIA.DEV Phase III. This overview illustrates the general approach of the third phase. Details of technology selection are abridged. The full process is depicted in Appendix B.

generated data needs to be made available to the system (cf. Figure 4.12). This connection guarantees that an acquisition technology can properly "dock" to incoming data sources, as storage is optional and analytics or processing activities can follow after that. An acquisition layer mapping can be omitted, if data is accessed, respectively stored directly without need for technical management (e. g., through streaming data flow management, cf. Section 2.2.6), pre-processing tasks such as complex data integration of heterogeneous sources or intensive data cleaning, which involves accessing, processing, and loading of the aforementioned.

Select Technology Classes. For each layer, appropriate technology classes are selected. Each class emits assigned tools from the GOBIA tool repository as alternatives for the customized BI architecture. Depending on layers, additional capability checks may filter unfitting tools immediately, e. g., if a BI functionality is based on classification, any analytics tools not supporting classification at all are excluded. As outlined above, the selection in each layer is based on several criteria, which resulted on the analyses from the previous sections. An overview of all technology selection criteria, which is discussed in the following, can be found in Table 4.13. One exception is the data generation layer, which explicates data sources and its properties to provide its information for the selection process in the acquisition layer.

> **Determine Generated Data.** This activity employs the same data dimensions of variety and velocity from the GOBIA.REF functional reference architecture to characterize the specific data source. Based on this characterization, its basic velocity properties are directly employed to drive acquisition technology selection.
>
> **Select Data Acquisition Technology (cf. Figure 4.13).** The key determinant in selecting acquisition technology is the velocity of data. For resting data, a number of mature and novel tools can be considered. These fall into the category of ETL and data integration tools. Here, traditional ETL suites, but also Big Data capable solutions such as Apache Flume are emitted. For streaming data, both streaming data collection tools as well as stream processing engines are activated to

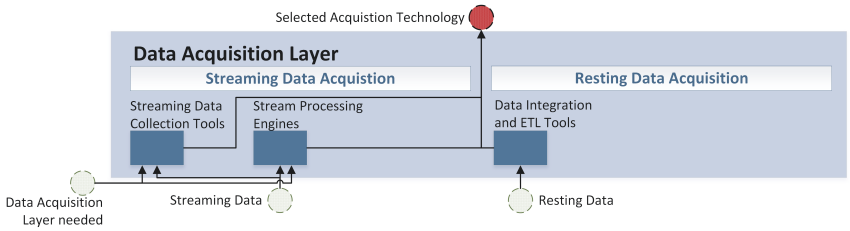

Figure 4.13: Extracted Petri net sub-process model of the overall GOBIA.-DEV Phase III process, which depicts the selection of technologies on the data acquisition layer.

emit their corresponding items. The reasoning is that processing incoming streaming data necessitates specific technology as technology for resting data acquisition, even when Big Data capable, is not specifically engineered for high data velocity. For instance, complex ETL workflows or MapReduce-based data integration introduce several latencies as they are designed for batch-style data ingestion.

Select Data Storage Technology (cf. Figure 4.14). Storage technology selection is driven by three main factors derivable from data properties. First, data volume dictates whether a distributed, horizontally scalable, system must be chosen. If data does not fit into a single server machine or small manually sharded cluster (e. g., as possible with traditional RDBMSs), scalable solutions are needed. If the data volume cannot be measured completely at this point, an expert measurement can be provided to decide if a single machine is sufficient for data storage. Importantly, this decision boundary shifts as technology evolves. Nowadays, single server storage capacities can exceed 100 TB of storage and 250 GB of main memory. For instance, Dell's PowerEdge R940 server offers up to 122 TB space and up to 384 GB of memory [158]. Naturally, actually available server capacities at an organization have to be considered. However, if data volumes are expected to exceed such sizes, a distributed solution is more suitable. Nevertheless, even if single server capacity is not exhausted completely at once, operating a

disk array or main memory at near maximum capacity may lead to performance drawbacks in the future (e. g., due to memory swapping, cf. [107]). Apart from that, it should be noted that distributed systems require additional efforts for coordination such as Apache YARN or ZooKeeper, which might increase complexity — especially, if a centralized solution is feasible as well. Additionally, data variety influences storage technology selection. In general, unstructured data such as video and audio files are well-suited for file systems. Both distributed and centralized file systems can store any data files as-is without changing their data structure or format. In particular, unstructured files that should be stored in raw format, would need additional preprocessing to force them into indexed storages such NoSQL data stores or RDBMSs.

Afterwards, the third determinant is applied by inspecting the format the input data to distinguish between a choice for distributed file system and the larger cluster of NoSQL data stores. For the former, which are represented by HDFS, large files are most advantageous, e. g., due to HDFS block size. Apart from that, main memory may become a bottleneck with many small files, unless these are aggregated into larger ones (cf. Section 2.2.3). This behavior could also be monitored in practical use cases (cf. Section 3.5). When the input data consists of moderately-sized or small data sizes, irrespective of volume, indexed storage is more suitable. Datasets focus on semi-structured data, but do not exclude file-based input. The property *moderately-sized* is to be seen in comparison to very large files in HDFS, which can be hundreds of MBs or several GBs. For example, document store MongoDB has an internal limit of 16 MB per document [282], while Redis has a value size limit of 512 MB [346]. Unless datasets of this or similar sizes can be extracted from files, distributed file systems as HDFS should be preferred. For structured distributed data, horizontally scalable SQL RDBMSs should be used as these fit the input data model best. A structured data format implies that no further data format inspection is necessary. Data volume is handled due to the system's property of being distributed. For centralized storage, the selection is entirely based on data variety. That

way, the most suitable store for the input data format can be chosen. While centralized file systems represent a feasible choice for unstructured data, structured data implies an RDBMS in the centralized case as well. However, for semi-structured data, still NoSQL data stores as horizontally scalable systems are recommended. In particular, these can also be run in smaller clusters or in standalone mode and offer a wide variety of storage options for schema-less data.

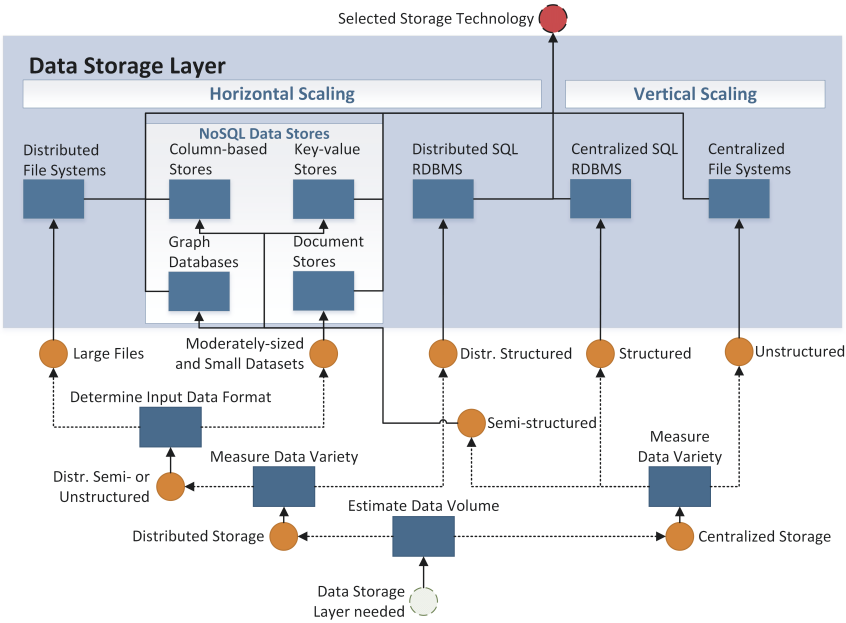

Figure 4.14: Extracted Petri net sub-process model of the overall GOBIA.-DEV Phase III process, which depicts the selection of technologies on the data storage layer.

Select Data Processing Technology (cf. Figure 4.15). For processing technology, the initial choice is between processing capabilities of a storage technology with in-database processing and dedicated process-

ing environments such as MapReduce and SPEs. Generally, databases and NoSQL data stores offer querying capabilities on indexed data. For example, RDBMSs offer SQL processing support and NoSQL data stores offer varying degrees of query-like processing support, which is mostly less capable than relational SQL. In practice, indexing capabilities of such databases are heavily used for this specific purpose. Twitter uses fast in-memory key-value stores to answer simple lookup queries quickly (cf. Section 3.5). Document store MongoDB offers richer querying capabilities into data structured in document-format, while SQL leverages strictly structured data formats. While indexed data can be queried efficiently, incoming data might need to be appropriately indexed before sophisticated queries are possible. While in-database processing can be either centralized or horizontally scalable, dedicated processing environments are geared towards MPP, which is especially suitable for generic processing of high-volume or high-velocity data. Thus, the first choice is regarding the *Processing Focus*, i. e., if *Indexed Data Querying* is preferred over *Massive Parallel Processing* or the other way round. However, choosing one of these does not mean that no parallel processing or some querying capabilities are not desired. As scalable indexed databases, such as distributed RDBMS, also allow for distributed query processing, this first choice underlines which capability is primarily needed to adequately represent the respective storage ABB. If indexed data is chosen, the next choice is between *SQL Processing* and *NoSQL Processing*. In comparison to querying capabilities of NoSQL data stores, SQL can be considered the more capable query language (cf. Section 2.1.1). Thus, it should be chosen if *Ad-hoc Queries* are needed as these can be flexible designed. This includes complex queries with various aggregate functions and joins. In contrast to that, *Simpler Queries* refers to queries which are generally less complex and powerful than complex SQL queries (e. g., regarding joins, aggregate functions, or OLAP extensions). As the querying functionalities of NoSQL data stores vary greatly, this term is chosen to summarize this fact. While document stores or graph databases might offer more enhanced querying

support, key-value stores such as Redis are less capable in that regard (e. g., only support retrieval of value associated with keys). Naturally, comparatively simple queries can also be formulated with SQL. Therefore, it is also an option if only simpler queries are needed. For an MPP processing focus, the choice of processing environment is determined by the *Analysis Latency Requirements*. These have to be extracted from the requirement specification. If this is not defined, the process cannot continue, unless this is amended. Notably, it is sufficient to define that, e. g., no real-time at all is needed — i. e., no specific target latencies are required. If no near real-time or real-time response time is expected, a *Batch Processing* solution such as MapReduce can be chosen. Alternatively, *Unified Processing* is also a viable choice, as these solutions rely on flexibly-sized batches and may cover changing data velocities (e. g., Apache Spark). For strict real-time responses, *Stream Processing* is selected, because it is specifically designed for such low latencies (cf. Section 2.2.6). The analysis in Section 3.5 further highlighted that low latency processing associated with real-time requirements warrants a stream processing solution, which primarily operate in-memory. Lastly, if relaxed *Near Real-Time* responses are necessary, *Unified Processing* is chosen in addition to *Stream Processing*. Micro-batch based stream processing might be able to process streaming data in general, but struggles with strict real-time responses.

Select Data Analytics Technology (cf. Figure 4.16). The selection of analytics is aligned to the structure of the data analytics layer in the GOBIA.REF technological reference architecture. In this layer, either technology for *Advanced Analytics* or *Traditional BI Analytics* is selected. As outlined in Section 4.2, the latter subsumes the functionality of complex *OLAP Functions*, while aggregate measures and descriptions as well as direct information access (cf. Figure 4.2) are handled as processing functionality. This leaves *OLAP Tools* as choice, when traditional BI analytics are set in the target BI functionality of the given analytics ABB. For all other cases, *Advanced Analytics* is the appropriate choice. Here, the *Analytical Workload* is to be estimated to decide, whether

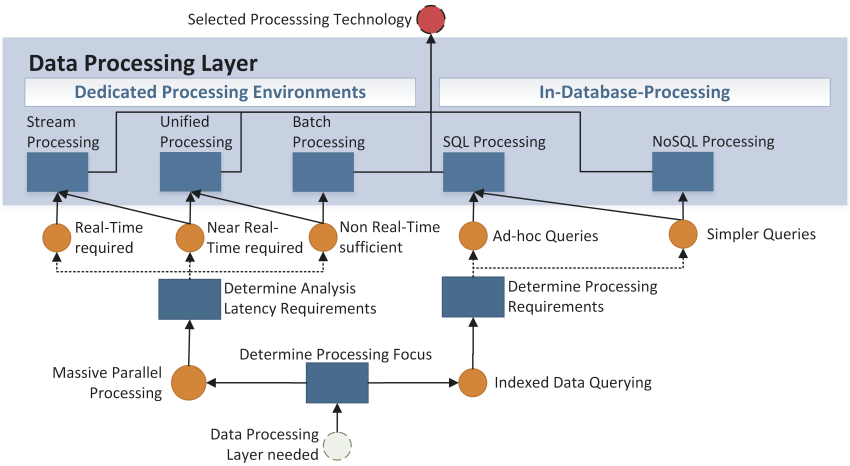

Figure 4.15: Extracted Petri net sub-process model of the overall GOBIA.-DEV Phase III process, which depicts the selection of technologies on the data processing layer.

Centralized Analytics on a single machine is sufficient and *Centralized Analytics Tools* are chosen or *Distributed Analytics* is needed, which leads to *Distributed Analytics Tools*. The approach for estimation is conceptually similar to the decision between distributed and centralized storage. However, an estimation is necessary, as pre-processed data might be reduced in size, which results in a different data volume for analytics. Moreover, not all portions of the data might be subject to analysis. For instance, if BI functionality is chosen so that a sentiment analysis is only conducted on a select number of data points from the current month, data volume can be significantly reduced. The available hardware as well as typical server and workstation capacities can be taken into account here as well. Notably, analytics workloads might require more main memory than data storage or general processing, depending on computations and chosen algorithms. In fact, optimizing memory efficiency and usage to increase performance is an ongoing

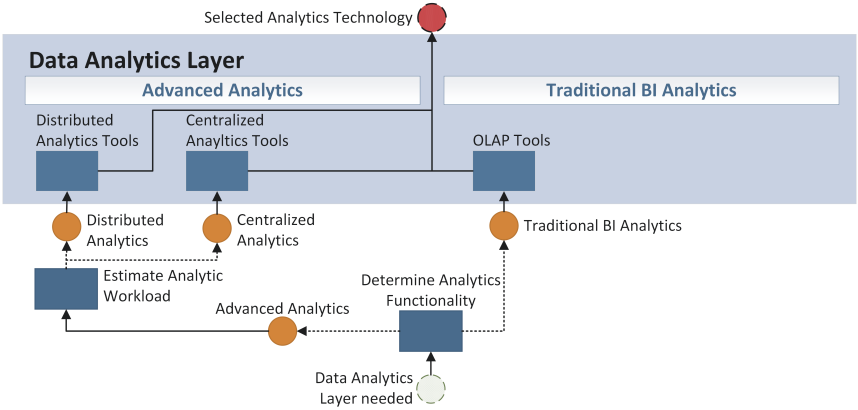

Figure 4.16: Extracted Petri net sub-process model of the overall GOBIA.-DEV Phase III process, which depicts the selection of technologies on the data analytics layer.

research topic, which is not just limited to centralized analytics (e. g., [ZR14, ZTCO14, Rei16]). Lastly, tools from the GOBIA tool repository are emitted according to the chosen class. However, tools that do not offer required BI functionality can be directly excluded by consulting the supplementary capability data in the repository. If only partial support is given, this decision can be postponed until the detailed technological evaluation of the tools in the post-GOBIA phase (cf. Section 4.4).

Create Compatible Technology Sets. After the selection of technologies has concluded for all ABBs, it has to be ensured that the selected technology alternatives are compatible. For instance, if a processing building block follows a storage block, a processing technology choice must be able to read data from the selected storage technology. For example, MapReduce and MySQL are compatible to each other, because MapReduce supports JDBC sources and other generic Hadoop input formats (cf. Section 2.2.4) — although a more detailed evaluation later might reveal that other combinations of storage and processing technologies exhibit more advantageous performance character-

Table 4.13: Decision criteria for technology selection in GOBIA.DEV Phase III. The criteria are listed for each layer and the decision alternatives, which can result from the decision are listed besides them.

Criteria	Layers	Decision alternatives
Data velocity	Data Acquisition	Resting data Streaming data
Data volume	Data Storage	Centralized Storage Distributed storage
Input data format	Data Storage	Large Files Moderately-sized and Small Datasets
Data variety[xxi]	Data Storage	Unstructured Semi-structured Structured
Processing focus	Data Processing	Indexed Data Querying Massive Parallel Processing
Analysis latency requirements	Data Processing	Real-time required Near real-time required Non real-time sufficient
Processing requirements	Data Processing	Ad-hoc queries Simpler queries
Analytics functionality	Data Analytics	Traditional BI Analytics Advanced Analytics
Estimated analytical workload	Data Analytics	Distributed Analytics Centralized Analytics

[xxi] Decision is separate for centralized and distributed storage.

istics, because MapReduce works best with distributed file system storage such as HDFS. In essence, each selected technology for an ABB must be compatible with the choices in each preceding and subsequent ABB. To determine compatibility between two tools, the compatibility matrix of the GOBIA tool repository located in Appendix C.2 can be consulted. As a result, this activity outputs sets of sequences of technologies according to the tactical plans of the semi-concrete BI architecture. In each of these sequences only one technology choice per ABB remains. For example if two compatible choices for both a storage (S_a, S_b) and a processing building (P_x, P_y) block remain, four sets of technologies are created: $\{S_a, P_x\}, \{S_a, P_y\}, \{S_b, P_x\}, \{S_b, P_y\}$. If no compatible sets of technologies remain, the activity *Assess Mitigation* is triggered. An expert assessment could lead to the conclusion that the GOBIA tool repository has to be extended, e. g., if novel yet unlisted tools emerge such as a new distributed file system more capable than HDFS. This might trigger further changes to GOBIA.DEV and GOBIA.REF if such new tools fall into new technology classes or shift the decision criteria in the layers of Phase III. This topic is further discussed in Section 4.5. The second possibility, as with the other two phases, is to refine the goals and restart at this or at an earlier phase to find a more suitable solution.

Compose BI Architecture Alternatives. Lastly, the compatible technology sets are used to create concrete and customized BI alternatives. For each technology set, technologies are structured according to the layers of the GOBIA.REF technological reference architecture, which were used for selection. The relationships between these technologies is derived from their source ABBs. Consequently, a similar architecture as in Phase II can be constructed (cf. Figure B.5). These given BI architectures represents viable solution alternatives for implementing a customized BI architecture. These filter the overall solution space significantly, which reduces complexity induced by both the technology landscape and its divergent patterns. The detailed evaluation of these BI architectures follows after GOBIA.DEV has concluded. This includes measuring required and achievable throughput and latencies as technological factors as well as the other three factors from the

TORE framework, e. g., organizational factors such as employee skills. This evaluation is elaborated upon as post-GOBIA activity in Section 4.4.

Application of Phase III to the FROG AIR Example

Decompose into Architecture Building Blocks and Assign Layers. In the first two steps of Phase III, each building block of the FROG AIR semi-concrete BI architecture is assigned to one or two of the technology layers from the GOBIA.REF technological reference architecture. As in the previous phase, the preparation chain is handled first and separately, because it is shared by all three tactical plans TP_{1-3}, which comprise the semi-concrete BI architecture. Furthermore, the ABBs representing the two BI data entities *Twitter messages* ($BIDE_1$) and *Facebook messages* ($BIDE_2$) are assigned to the data generation layer in order to aid selection of potential acquisition technologies. Notably, all information which the semi-concrete BI architecture was built upon is potentially needed. This applies especially to the formulated goals and requirement of the FROG AIR case.

Select Technology Classes. Based on Figure 4.10, the following assignments are made:

The generation ABBs for *Twitter messages* 1.0 ($BIDE_1$) and *Facebook messages* 2.0 ($BIDE_2$) naturally lead to the data generation layer. Both data are semi-structured **resting data**. This information influences the selection of a data acquisition technology class in the next step and, indirectly, the data storage selection.

For the corresponding storage building blocks 1.1 and 2.1, acquisition and then storage technology classes are to be selected. An acquisition technology is warranted, as these can actively load JSON data sources from the Internet and ensure that these documents can be loaded by the subsequent storage technology. With both data sources being resting data, the choice falls into the category of *Resting Acquisition* and, more specifically, to **Data Integration and ETL tools**. The tool repository contains *Pentaho Data Integration (Kettle)*, *Talend Open Studio*, *Apache Sqoop* as solution alternatives for this choice. As the subsequent building block is data storage, the required

capabilities include to access a JSON source from the Internet and load it into storage, where additional processing blocks are applied. Notably, this only keeps *Pentaho Data Integration (Kettle)* and *Talend Open Studio* in the selection, as Sqoop does not support loading JSON files from the Web.

To select storage technology in the data storage layer, several questions regarding the data need to be answered. First, the overall data volume is measured or estimated to decide if a distributed storage system is necessary or if a single system (centralized) setup is sufficient. While Twitter and Facebook have a huge corpus of data overall, the data of interest can be considered significantly smaller, because only a limited set of Twitter accounts and Facebook pages is considered. As the selected subset of the sources in question does not contain millions of posts and posts are only textual data, a *centralized storage* is sufficient. To choose between *Centralized SQL RDBMSs*, NoSQL data stores, and *Centralized File Systems*, data variety is employed as discriminator. As the original source data is semi-structured, **NoSQL data stores** are a natural choice. From this broader class *Document Stores* are selected, because JSON text results from Facebook and Twitter APIs resemble documents. The tools *MongoDB* and *Couchbase* are emitted from the repository as alternatives as both are able to work with JSON data. Alternatively, a transformation into a structured data format and the use of a *Centralized SQL RDBMSs* could have been possible. Typical *Data Integration and ETL Tools* often offer needed functionality for such conversions readily. However, there is neither a functional nor a non-functional requirement referring to such additional data transformation.

For both the message filtering and joining activities (1.2, 2.2, 1.3) a data processing technology is needed. As the focus of the BI functionalities encoded in the ABBs of the semi-concrete BI architecture is on querying information (*Indexed Data Querying*) and MPP is deemphasized due to the overall data volume, the specific processing complexity requirements dictate the technology class to be employed next. TP_1 and TP_2 do not need complex querying capabilities and only limited ad-hoc queries to perform necessary filtering and data combination. While joins for combining data can become complex, the limited join capabilities of document stores such as *MongoDB*

and *Couchbase* are considered sufficient for this case. This can be derived directly by consulting the capability matrix in Appendix C. Moreover, these joins are not executed repeatedly, but only once during data preparation after storage. TP_3 will need an advanced analytics technology from the data analytics layer and does only need simple querying capabilities to pass this data to that layer. In this case, both conditions can be satisfied by **NoSQL Processing**, which is done in-database. While dedicated processing environments could theoretically be applicable, it is not necessary in this case, because in-database processing is also a better fit to the *NoSQL data store* choice from the previous step.

For the final processing building blocks 2.5 and 3.5 from TP_2 and TP_3, which represent the BI functionality to be implemented, also **NoSQL Processing** is chosen. The criteria are the same as in the selection before. *Counting* and *average* calculation are found in several NoSQL data stores, in particular document stores, and are rather simple aggregation activities. In spite of that, due to the heterogeneity of NoSQL processing capabilities (in contrast to SQL processing), special consideration must be laid on the actual product selection later on to ensure that specific processing capabilities are met (e. g., if aggregate or join functions in a particular NoSQL data store are available). In this case, consulting the capability matrix of the tool repository and listed vendor documentation sources provide sufficient information to determine that the partial aggregation and join functionalities are sufficient. Otherwise, a detailed technological evaluation afterwards in Section 4.4 would be necessary.

The three main BI functionalities in tactical plans TP_1, TP_2, and TP_3 have different requirements on the data analytics layer. While TP_2 and TP_3 employ *Aggregate Measures & Descriptions* as BI functionality that can be handled by NoSQL processing capabilities. However, the sentiment analysis in TP_1 necessitates a technology from the data analytics layer. Sentiment analysis is mapped to *Advanced Analytics*. To choose a fitting technology class for that, the decision activity *Estimate Analytical Workload* is conducted to specify whether rather a distributed one or a non-distributed one is suitable. As the data size is limited and no real-time analysis is needed, *Cen-*

tralized Analytics is appropriate. Therefore, **Centralized Analytics Tools** are selected. For these, *Anaconda (R & Python)*, *RapidMiner (Standalone)*, *KNIME*, and *Weka (Standalone)* are emitted from the GOBIA tool repository. All of these tools have a direct capability to conduct sentiment analysis (cf. Table C.4), albeit KNIME marks this as "beta" feature. Subsequently, the analysis results must be stored as specified by building block 1.6. As before, delivering sentiment analysis results focuses on indexed querying instead of MPP. The data variety of analysis results depends on the output of the analytics tools. A sentiment analysis is a simple data format. It basically consists of a message ID, a message text, and a determined sentiment. This data structured can be exported by targeted analytics tools either as semi-structured file (e. g., CSV, JSON, or XML) or written into a database. In addition to that, RapidMiner supports a direct export into MongoDB (cf. Table C.4). Thus, a semi-structured *JSON* export is selected, which allows to re-use the previously selected technology class of **NoSQL data stores** and document stores. The same set of tools as before is emitted.

Additionally, a separate storage building block 1.4 was defined to store the joined Facebook and Twitter messages. This is the building block that provides the data to the final processing and analytics ABBs. In this case, no technology change is necessary, as the criteria are similar as before. Therefore, **NoSQL data stores** are selected. Also, both the preceding and succeeding processing building blocks yield the same **NoSQL Processing** choice.

An overview of all choices is summarized in Table 4.14.

Create Compatible Technology Sets. After technologies have been selected and capable tools are emitted, compatibility between these tools must be ensured. In the FROG AIR case, NoSQL data stores, NoSQL processing, and centralized analytics tools are chosen.

Naturally, NoSQL processing is compatible to NoSQL data stores. Thus, no compatibility issues remain in this case. Document stores can easily import JSON documents and data without further modification. Therefore, both *MongoDB* and *Couchbase* are kept in consideration.

Table 4.14: Technology classes selected for FROG AIR in GOBIA.DEV Phase III.

Tactical plan	Element	Layer	Technology Class Selection
TP_{1-3}	(1.0) Generation	Data Acquisition	Data Integration and ETL Tools
TP_{1-3}	(1.1) Storage	Data Storage	NoSQL data stores
TP_{1-3}	(2.0) Generation	Data Acquisition	Data Integration and ETL Tools
TP_{1-3}	(2.1) Storage	Data Storage	NoSQL data stores
TP_{1-3}	(1.2) Processing	Data Processing	NoSQL Processing
TP_{1-3}	(2.2) Processing	Data Processing	NoSQL Processing
TP_{1-3}	(1.3) Processing	Data Processing	NoSQL Processing
TP_{1-3}	(1.4) Storage	Data Storage	NoSQL data stores
TP_1	(1.5) Analytics	Data Analytics	Centralized Analytics Tools
TP_1	(1.6) Storage	Data Storage	NoSQL data stores
TP_2	(2.5) Processing	Data Processing	NoSQL Processing
TP_2	(3.5) Processing	Data Processing	NoSQL Processing

From a strict compatibility point of view, *Talend Open Studio* is the only ETL tool, which can directly connect to both MongoDB and Couchbase. Apart from that, both Talend Open Studio and Pentaho Data Integration are able to query JSON documents from the Internet. However, Pentaho cannot access MongoDB directly, i. e., data has to be exported and then imported into MongoDB by other means. This is not covered by Apache Sqoop.

RapidMiner can directly access MongoDB using a dedicated NoSQL extension. KNIME is able to do this as well, but the extension is still marked as experimental and thus only partial supported is given. The other tools need intermediate exports and manual data imports and are thus marked as “incompatible”. Arguably, the direct connection between RapidMiner, KNIME and MongoDB is most efficient, as the other tool combinations need intermediary JSON files, which need to be ex- and imported. Nevertheless, a

solution with intermediary files can be taken into consideration again, if the selections made here should be deemed not suitable later in the process. As both analytics solutions are most compatible with MongoDB, it remains as sole choice for data storage.

For data acquisition, **Talend Open Studio** is a fitting choice. It is compatible to a wide range of databases and MongoDB as well Couchbase in particular. Additionally, Talend supports fetching JSON documents from the Web. *Pentaho Data Integration* is directly compatible with MongoDB, but not directly with Couchbase as a manual JSON export/import has to be employed. Thus, both tools can remain as being generally compatible.

The compatible technology mix as determined by GOBIA.DEV is therefore: **(Talend Open Studio, Pentaho Data Integration)**, **MongoDB**, **(RapidMiner, KNIME)**.

Compose BI Architecture Alternatives. Lastly, compatible technology sets from before are compiled into several BI architecture alternatives. Figure 4.17 depicts a simplified combination of all alternatives using a Petri net.

4.4 Post-GOBIA Activities

Offered a set of viable, concrete BI architecture alternatives, an organization needs to assess which one is the most suitable alternative. Although GOBIA.DEV incorporates several factors directly or indirectly in its result, a detailed post-GOBIA evaluation of the alternatives can regard aspects that were out of scope of the GOBIA method. For example, inviting vendors to conduct product demonstrations or evaluating and benchmarking prototypes in extensive details are typical activities for software selection processes (e. g., cf. [Sta03, pp. 231ff.], [RGS02, pp. 107ff.]). Some of these criteria have been pictured previously in Section 3.6 such as performance. In this subsection, an illustrative selection of criteria and points of consideration are presented according to the analysis perspectives provided by the TORE framework. A generic process for the evaluation of alternatives *post-GOBIA* is depicted in Figure 4.18.

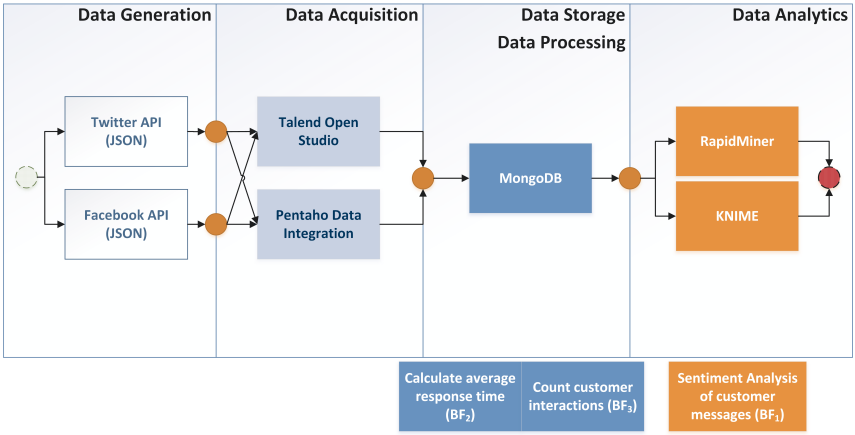

Figure 4.17: GOBIA.DEV Phase III for the FROG AIR case: Combination of all technology alternatives on all layers of Phase III. Three data uses (BF_{1-3}) are shown below the model. Processing and Storage functionality are shown as one layer, because the same technologies are used. Data flows are not depicted separately.

Technological Evaluation

One criterion that was considered in Section 3.6, but was no explicit part of the model, was *performance*. While *latency* was considered during processing technology selection, throughput performance was simplified to a selection between an MPP and indexed access focus. To measure throughput and latencies accurately, benchmarks are needed. These are experimental evaluations of the performance aspects of software. Although some approaches for benchmarking Big Data solutions exist (e. g., [GRH+13]), benchmarking a specific set of architecture alternatives involving several pieces of software, is still challenging.

Several attributes, which are listed as *supplementary attributes* in the GOBIA tool repository (cf. Section C) constitute more detailed and tool-specific criteria, which did not materialize as generalized decision criteria

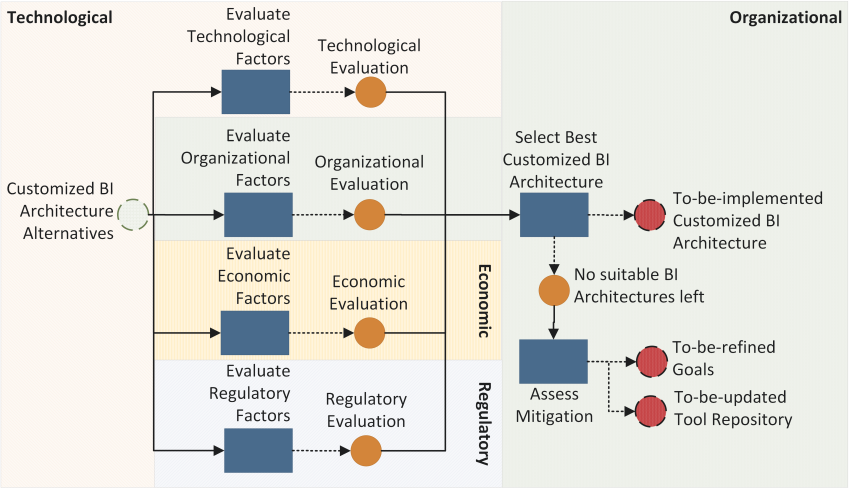

Figure 4.18: The evaluation of alternatives as post-GOBIA process depicts the assessment of the customized BI architecture alternatives according to the perspectives and factors provided by the TORE framework.

in GOBIA.DEV Phase III. For instance, specific querying capabilities, which were only generally considered here, or a focus with respect to the CAP properties could be used for further evaluation.

Furthermore, the *complexity* of the overall solution is to be considered. As BI architecture alternatives can consist of a few tools, which are applied to several tasks or a larger set of tools, each for one task only, has impacts on the evaluation of architecture, which need to be considered.

Naturally, besides the technical criteria outlined in Section 3.6 and that became part of the method, other generic properties of systems can be considered. For instance, Pohl lists *non-functional requirements*, which are quality criteria with respect to the system in general (cf. Section 2.5). These include capabilities such availability, which denotes the handling of system faults and how long it needs to stay reachable when these faults occur.

Additionally, Starinsky specifies technical constraints such as constraints regarding the required operating system (e. g., an architecture should run on CentOS machines), interface (e. g., requiring a Web interfaces) or specific tools that are fixed in the architecture (e. g., because MySQL is already used company wide) (cf. [Sta03, p. 254]).

Organizational, Regulatory, and Economic Evaluation

This section provides an illustrative selection of organizational, regulatory, and economical criteria according to the TORE framework to consider for software selection. These are not necessarily specific to BI architectures, but can be applied to other software as well.

Organizational Evaluation. Organizational criteria relate to the employees and the structure of the organization conducting the evaluation. For example, for a given set of BI architecture alternatives, employees with the skills necessary to install, maintain, or use the software might be needed. In a 2011 study by Russom, issues while staffing Big Data analytics projects were considered the top adoption barrier to Big Data [Rus11, p. 12]. Similarly, a lack of business sponsorship, i. e., management level support for Big Data solutions, is named as an issue. Another less tangible aspect is the cultural fit of specific pieces of technology to its users. Nevo and Wade postulate that IT artifacts can only be effectively used in an organization for a competitive advantage when, among other factors, they are "compatible" with the employees using it. This is the case when a relationship between an employee organizational resource and the IT in question can be formed (cf. [NW11, p. 405].

Regulatory Evaluation. From a regulatory perspective, internal and external regulations can be distinguished. Internal regulations include, e. g., compliance rules or security regulations. External laws can influence software selection on many levels. For instance, the *General Data Protection Regulation (GDPR)* [169] in the EU might influence usage of personal data for analyses, e. g., if those persons are minors [123].

Economic Evaluation. Naturally, economic criteria include *cost*-related aspects for an architecture, e. g., such as direct costs of purchase and maintenance costs. Commercial software, e. g., SAPA HANA, can exhibit various licensing models, which influence total costs of an architecture [357]. Despite the fact that several Big Data and Streaming Processing Technologies such as the various Apache projects are open source and free-of-charge, other cost drivers might negatively impact cost-benefit calculations. For instance, the lack of staffing and employee skills mentioned above might incur cost for training or additional staffing (cf. [Rus11, p. 12]). A set of many different tool alternatives could further worsen this issue as several pieces of different software need to be maintained by appropriately skilled personnel.

4.5 Extending the GOBIA Method

Naturally, the GOBIA method may need to be extended to accommodate new developments in the realm BI architectures or to include current tools, which are not part of the initial tool repository. It is relevant to document both extension points and the process of extensions, so that the GOBIA method remains up-to-date and applicable in light of aforementioned changes. Moreover, organizations could extend or customize the GOBIA method to accommodate their specific needs.

The focus here is on internally consistent extensions instead of discussing changes in conceivable contingencies. This means that extension areas documented here add new elements or modify exist ones, as long as the underlying principles of the GOBIA method are not altered. For instance, adding a new, 4^{th} element type to the GOBIA.REF functional reference architecture would have far reaching implications on multiple design decisions and inner workings of GOBIA.DEV, which would necessitate extended re-engineering, which is out of scope here. However, adding, e. g., a new BI functionality element would not alter the basic principles of the development process.

In this section, three main extension areas are described. First, updating the GOBIA tool repository with either new tool or new attributes is illustrated in Section 4.5.1. Next, the extension of GOBIA.REF is outlined in Section 4.5.2.

The description concludes with the extension of GOBIA.DEV detailed in Section 4.5.3.

4.5.1 Updating the GOBIA Tool Repository

The GOBIA tool repository in Appendix C contains all tools categorized into technology layers according to their respective class. This attribute and other capability attributes are purpose-fitted to the elements of GOBIA.-REF and contain decision values for GOBIA.DEV Phase III. Additionally, the tool repository features a selection of supplementary attributes, which can aid post-GOBIA activities and provide more context to the respective tools. Two extension processes are described, adding new tools to repository and adding new attributes. After describing each process in general, an illustrative example follows.

Adding new Tools to the Repository

Adding a new tool to the existing repository under the assumption that the repository does not need to be extended consists of the following steps:

1. **Determine target layer and technology class.** The lists of the repository are separated by layers according to the GOBIA.REF technological reference architecture. A new tool can be placed in any layer except the data generation layer. The technology classes inside each layer function as selectors on a corresponding attribute of the target tool list. For example, when a centralized RDBMS is activated in GOBIA.DEV Phase III all tools are returned whose attribute *Technology Subclass* equals *Centralized RDBMS*. The technology class and, if applicable, subclass are also chosen according to the defined blocks in the GOBIA.REF technological reference architecture. Notably, for storage technologies, no extensive capability filters are applied. This is different for analytics tools, where the BI functionality serves as additional filter on the analytics tool list's capability matrix (cf. Section C.1.4), when emitting the tools during GOBIA.DEV Phase III.

2. **Determine necessary and supplementary attribute values.** After the target tool list has been identified, the individual attribute values of that list need to be supplied. All lists have a header containing the name of the tool, and basic attributes regarding examined version and release date of that version as well attributes denoting their technology class, subclass or other discriminative category, e. g., *Advanced* vs. *Traditional BI* as *Primary Category* for analytics tools in Section C.1.4. Supplementary attributes are those attributes, which are not used directly during GOBIA.DEV, but can employed in post-GOBIA activities (e. g., Indexing capabilities of databases, cf. Section C.1.2). These are marked in *italics*. The attribute value can be determined using publicly available vendor documentations or third-party evaluations. When using the latter, it has to be verified that the statements by third parties are still accurate. Due to quick development in the Big Data landscape, certain information can become outdated. For example, Apache Mahout deprecated its MapReduce engine in favor of others such as Apache Spark [16]. However, several analytics methods are not yet ported from the MapReduce engine to newer ones. The analysis of analytics capabilities by [LKRH15] still lists algorithms for Mahout, which are not supported any longer. In addition to that, an empirical evaluation of a tool can yield more information regarding its characteristics.

3. **Determine capabilities for capability matrix.** In addition to textual fields among the regular attributes, the tool lists contain a capability matrix using ✔, ✖, and (✔) (cf. Section C.1) as visual markers. For example, the list of storage technologies provides a capability matrix, which aids in determining if a storage solution supports the desired querying functionalities. The evaluation of not directly deductible capabilities and other attributes is explained at the beginning of each list in Appendix C.

4. **Determine compatibility to other tools for compatibility matrix.** Finally, the compatibility between the to-be-added tool and the

other tools in the tool repository needs to be determined and documented in the compatibility matrix in Section C.2. The matrix lists "ingoing" and "outgoing" compatibilities. For each row, the outgoing compatibility to each tool in the columns is set. Outgoing compatibility from a tool *A* in a row to a tool *X* in a column is given if data can be transferred from tool *A* to *X* — or if tool *X* can access data from tool *A*. Ingoing compatibility is thus read for each column and denotes if tool *X* can have data from tool *A* as input. For instance, *Talend Open Studio* has both an *Apache Kafka* reader and a writer component, which means that both ingoing and outgoing compatibility is given (cf. Table C.5). Section C.2 lists the criteria for full ✔, partial (✔), and no compatibility ✖. In short, ✔ full compatibility means that transfer can directly happen between two tools without limitations. (✔) partially compatible tools possess some form of limitations like specific conditions, which need to be met for data to be transferred or if a tool has only "beta" level support for another tool. ✖ denotes no direct compatibility between tools, which means that either custom application level connectors and code need to be written or that another tool needs to be used as proxy in between, e. g., a centralized file system. If the documented tool is an extension and relies on the compatibility of its platform, compatibility needs only to be denoted to its platform, e. g., as done for Apache Kafka Streams (cf. Section C.2).

Example: Adding Vertica. Vertica is a distributed RDBMS, which uses a columnar data layout. It is positioned for OLAP and DWH use (cf. Section 3.3).

Basic information is sourced from the Vertica website and its documentation. In this case, Vertica is rooted in a scientific project called *C-Store* and respective academic papers describe its inner workings in detail [BLT12] [LFV+12] describing its inner workings in detail. Thus, it can be confirmed that Vertica can be considered a *Distributed RDBMS*. Technology class, subclass, and primary storage location can be deduced from aforementioned publications. Notably, this information is also supplied in the documentation (e. g. [260]). To assess the focus of Vertica with respect to the CAP theorem, it is determined if it provides support for *ACID* guarantees, which is the case

[259]. As Vertica features a distributed shared-nothing architecture for its cluster [BLT12], *CP* can be set as CAP focus attribute value. Notably, this attribute is a supplementary property not directly used in GOBIA.DEV. The list of attribute values and their origin is listed in Table 4.15.

Table 4.15: Properties of Vertica for illustrating how to add it into the GOBIA Tool Repository for storage tools. Attributes in *italics* denote supplementary attributes.

Attribute	Vertica	Source
Version, Date	9.0.1-7, Apr 2018	[256]
Technology Class	RDBMS	[BLT12]
Technology Subclass	Distributed RDBMS	[BLT12]
Primary Storage	Disk	[BLT12, LFV+12]
CAP Focus	CP	[BLT12, 259]
Access Interface	SQL	[258]
Query: Projection	✔	[258]
Query: Selection	✔	[258]
Query: Aggregation	✔	[258]
Query: Joins	✔	[258]
Adv. Query: Subqueries	✔	[258]
Adv. Query: OLAP functions	✔	[258]
Indexing	(✔)[xxii]	[258, 257]

[xxii] Text-index available [258]; uses columnar projections instead of generic indexes [257].

With respect to an *Access Interface*, Vertica offers full *SQL* compatibility so that *SQL* can be set for the attribute. The *Access Interface* attribute is evaluated during technology selection in GOBIA.DEV Phase III. Therefore, Section C.1.2 provides a list of possible values so that a known set of attribute values is present during selection. For the supplementary values regarding query feature and indexing support, the SQL reference in Vertica's documentation can confirm that there are no omissions to most SQL features. However, indexes are limited by design in Vertica. Instead of most indexes, Vertica

uses *projections* utilizing its columnar layout [260]. Although these can be user-defined as indexes, the strong discouragement in the documentation [257] leads to this supplementary attribute being evaluated as "partial" (✔). A table footnote is amended to add further textual information.

Finally, the **compatibility** to the other tools of the repository is tested and the compatibility matrix is filled out. If two tools can work together immediately with immediately available building blocks full compatibility is denoted. To assess compatibility, not only the Vertica documentation needs to be consulted, but the sources and documentations of the other tools of the repository as well. Particularly for RDBMSs, no direct support for a specific tool is needed, but can be covered by generic JDBC or ODBC, which allow several tools to ingest data any database, which supports these interfaces.

As no tool has special limitations with respect to Vertica[6], the compatibility documentation can be derived from other RDBMSs such as MySQL or PostgreSQL. Notably, unique support of these has to be excluded such as specific support for PostgreSQL for Apache MADlib [34].

Adding new Attributes to the Repository

In addition to adding a new tool, it might be necessary to add new attributes to the repository. The process is similar to updating existing attributes, which can be interpreted as replacing an old attribute with a new one.

1. **Determine new attribute.** First, a new attribute including its target technology layers have to be identified. For instance, a new attribute can emerge as new decision criteria for an updated GOBIA.DEV Phase III (cf. Section 4.5.3). Another source of attributes are additional information on post-GOBIA evaluations, where supplementary data about tools is useful. For instance, if a unified throughput benchmark for a certain tool category is available, the results might be used during technological evaluation. Tool licenses and associated costs could be considered during economical evaluation. Any attributes mentioned

[6] For instance, Microsoft SSAS limits multidimensional cubes to selected databases such as Oracle [270].

in Section 4.4 might become a candidate for permanent inclusion in the repository. The target technology of an attribute determines in which tool lists an attribute is added. While universal attributes such as *Version, Date* are found in each tool list, some attributes are only useful for some or a particular technology layer and its tool list (e. g., support for classification algorithms is only used for data analytics tool, cf. Section C.1.4).

2. **Determine attribute value range.** After an attribute has been chosen, its to-be-documented value range has to be decided upon. It should be depending on utility for GOBIA.DEV or desired expressiveness as supplementary attribute. For instance, the attribute *Access Interface* for databases needs to aid in indicating whether complex ad-hoc queries are supported like in SQL processing or only simpler queries as in NoSQL processing. While a binary classification would certainly be sufficient for GOBIA.DEV Phase III alone (e. g., *SQL support:* ✔/✖), the specific querying capabilities are needed for a detailed in evaluation post-GOBIA. Here, it needs to be verified if the specific BI functionality — e. g., a particular aggregate function such as *ranking* — is supported. Thus, it is purposeful to already use a more detailed categorization for *Access Interface*. Here, *SQL*, *SQL-like*, or *API* are used as value range. SQL-like languages indicate support for a subset of the querying capabilities of SQL, while APIs offer programmatic and imperative access to functionality. Additionally, selected query language capabilities were chosen as supplementary attributes (cf. Section C.1.2). Possible value types include:

   Binary or Ternary. A binary attribute ✔/ ✖ indicates absence or availability of the selected attribute at the tool in question and is employed to set up the *capability matrix* for a tool. Partial availability (✔) is denoted with a ternary attribute, which states that an attribute is generally available for tool, but limitations exist that inhibits its full utilization. For instance, for *SQL support* a trinary attribute can be used to denote full ✔, partial (✔), or no

✖ SQL capabilities. In any case, an additional documentation for that attribute should reveal, which criteria lead to an attribute to be categorized as ✔, (✔), or ✖. For instance, a less capable SQL-like query language could be denoted with *SQL support: (✔)*. In simple cases, a self-categorization of tool vendors for offering *SQL-like* functionality could be used to assume only partial (✔) SQL support. For a more detailed evaluation, it might additionally be argued *why* these languages offer less features than SQL. This might be done by, e. g., listing possible sub-features to be identified in the documentation or in a standard specification (e. g., [ISO92]). For the GOBIA tool repository, several ternary supplementary attributes are used to denote aggregation, join, selection, and projection capabilities (cf. Section 2.1.1). This enabled to use a *categorical list* attribute for the *Access Interface* attribute as the supplementary attributes add further information on a particular access or query interface.

Categorial. A categorical attribute uses a predefined set of possible text values to assess an attribute. For instance, the basic values for *Access Interface* are categorical. Also, technology classes, or modes of operations, which can be either *Distributed* or *Centralized*, represent categorical values. Categorical values allow to "select" appropriate tools based on these attributes during GOBIA.DEV Phase III.

Free text. Free text attributes contain arbitrary textual values, which may follow a certain pattern (e. g., "Version 1.0, Apr 2018" for a *Version, Date* attribute) or be formulated freely. Such attributes require additional details on interpretation when they should be used for technology selection in GOBIA.DEV Phase III or for a detailed post-GOBIA evaluation.

List. Attribute lists combine several attribute values, separated by a list delimiter (e. g., a comma). The separated values can have any of the aforementioned value types. For instance, *Access Interface*

is a list of categorical values. Notably, a list can also have one entry only. A new attribute based on a list could be, e. g., *Classification algorithms*, where a list of free text names of classification algorithms of an analytics tool is provided.

3. **Evaluate attribute values for each tool.** Finally, for each tool in the target technology layer of the attribute, the attribute values have to be determined. This is done using the guidelines for assessment created while determining the attribute value range. The process is the same as when adding a new tool, except that only one new attribute is assessed (see above).

Table 4.16: Result of adding an attribute that indicates support for neural networks for the tool list for the data analytics layer in the GOBIA tool repository.

Tool	Anaconda	RapidMiner	KNIME	Weka	RapidMiner (Radoop)	Apache Mahout	Spark MLlib	FlinkML	Apache Samoa	Apache MADlib	H2O ML	Pentaho Mondrian	Microsoft SSAS
Neural Networks	✔	(✔)[xxiii]	✔	✔	(✔)	✖	✔	✖	✖	✔	✔	✖	✔
Sources	[161, 328]	[339]	[231]	[241]	[339]	[28]	[92]	[54]	[11]	[34]	[194]	[206, 323]	[266]

[xxiii] Using H2O.

Example: Adding a "Neural Networks" attribute for analytics tools. This example aims to add a new supplementary capability attribute to tools on the data analytics layer. *Neural Networks* should indicate if the tool supports any kind of neural network for either classification or regression. As

this is a specific technique for conducting aforementioned methods, this is a supplementary attribute. A *ternary* attribute is chosen to assess this attribute. While the functional support is indeed binary, as either building a specific neural network is possible "out-of-the-box" or not, several factors can contribute to partial (✔) support. For instance, neural networks could be still labeled as incomplete, experimental, or "beta" functionality (as done with sentiment analysis for KNIME [231]). For experimental or incomplete functions, utility may be limited, e. g., due to bugs or other malfunctions. In this case, RapidMiner offers support for neural networks, but only when using its H2O integration. Thus, its support is marked as partial (✔). Support for this feature is noted as ✔ if the listed sources for the tool in question indicate support for at least one neural network such as multi-layer perceptrons [HKP11, pp. 261ff.]. These are in fact supported by most tools in the repository if support was given at all, e. g., by KNIME [231], Weka [241], Spark MLlib [92], or H2O ML [194]. The results for the existing analytics tools in the repository and the used Web sources are listed in Table 4.16.

4.5.2 Updating GOBIA.REF

A part of the GOBIA method, which can be extended, are the GOBIA.REF functional reference architecture and the GOBIA.REF technological reference architecture. Although these are embedded into GOBIA.DEV, their structure can be discussed separately from GOBIA.DEV as done in Section 4.2. An update to GOBIA.REF means changing the architecture, i. e., the elements and relationships therein.

Functional Reference Architecture: Adding new BI Functionalities and Data Preparation Steps

The functional reference architecture is a layered architecture, which consists of BI functionalities, data preparation steps, and BI data entities. Inside each of those element types, blocks for the respective elements can be found such as "classification", "filter", or "semi-structured BIDE". BI functionalities are further distinguished into groups of predictive, descriptive, and non-analytics

BI functionalities. BIDEs are structured by variety and velocity and are *not* considered here, as no specific data entities are listed in the functional model and the characterization with the two aforementioned dimensions provides necessary input for decisions in GOBIA.DEV Phase III.

1. **Determine new element and add to model.** First, a new element needs to be identified. For instance, a new **BI functionality** could include new types of methods not included in the model yet. This can also be specialized forms of already existing elements as topic modeling or time-series analyses are, which are specific cases of regression or classification. For example, if *Video object detection* should become important enough to be deemed as distinct BI functionality, the element can be added to the *Descriptive Analytics* block. Similarly, if new **data preparation** steps are needed, they can simply be added to the element type block in the GOBIA.REF functional reference architecture. However, these need to be selected so that they resemble such preparation functionalities, which are placed in between data entities and functionality. The elements need to be described and discriminated from other elements to prevent overlapping, which introduce unnecessary ambiguities (as done in Section 4.2.1).

2. **Update GOBIA.DEV.** To be able to fully utilize a new element, the GOBIA.DEV process has to be updated as well, which is outlined cf. Section 4.5.3. Notably, several phases can be affected by new BI functionality or data preparation steps.

Technological Reference Architecture: Adding new Technology Classes

The GOBIA.REF technological reference architecture is structured into layers and technology classes, which drive technology selection and result in tools emitted from the GOBIA tool repository in GOBIA.DEV Phase III. As technologies evolve, the technology class to categorize them might evolve. Notably, an evolution which alters selection criteria is discussed in the following Section 4.5.3.

1. **Determine new technology class and target layer.** First, a new technology class needs to be identified, respectively specified. In GOBIA.REF classes were derived based on built-up knowledge throughout this work (cf. Chapter 2) and a brief overview of respective tools (cf. Section 4.2). For example, RDBMSs and NoSQL data stores are established technology classes. As subclasses of NoSQL data stores, four main classes were identified in Section 2.2.2. If in the future, hypothetically, the briefly described group of *Multi-model Databases* (cf. Section 2.2.2) were chosen to be added as another class, its *purpose* of data storage would direct it into the data storage layer. Apart from that, dedicated processing tools should be assigned to the data processing layer, if these are neither databases nor are used for data ingestion as streaming tools are.

2. **Update GOBIA.DEV Phase III.** When a new technology class is added to a layer, selection criteria in GOBIA.DEV Phase III might need to be updated (cf. Section 4.5.3).

3. **Update tool repository.** Finally, after a new technology class has been added, the tool repository must be updated to accommodate this class. At least the attribute that drives class membership (e. g., *Technology Class*) is to be adapted. However, if new selection criteria are necessary, the tool attributes might need to be added (cf. Section 4.5.1).

4.5.3 Updating GOBIA.DEV

After the structural updates to GOBIA.REF are concluded, the final part of extending the GOBIA method is to update the logic (i. e., the decisions) inside the GOBIA.DEV process. This sub-section gives a brief overview of selected extension points in GOBIA.DEV.

Phase I

Modifying Element Type Mapping. To enable the mapping from functional requirements to a new BI functionality, it must be clarified or illustrated

how a requirement is mapped to such an element. Here, a sample expression table is provided to aid mapping (cf. Table 4.5). Moreover, the logic behind the mapping is explained and illustrated using a running example and other case applications (cf. Chapter 5). These methods can be used to accommodate new BI functionalities. This approach is mirrored for data preparation steps.

Phase II

Modifying Architecture Building Block Mapping. GOBIA.DEV Phase II offers guidance and recommendations on mapping BI functionalities, data preparation steps, and BI data entities to appropriate architecture building blocks. For instance, *Aggregate Measures & Descriptions* is mapped to a processing ABB and several others to analytics building blocks. These recommendations can be updated if new BI functionalities are added or if these conceptual functionalities are implemented by a different type of technical functionality.

For instance, *Aggregate Measures & Descriptions* is deliberately mapped to processing functionality as it can be covered by respective technologies at the data processing layer. In case of the example from Section 4.5.2 — new BI functionality *Video object detection* — a general mapping advice to *analytics* ABBs might be recommended, possibly combined with additional processing blocks as needed to pre-process the video or conduct sampling. However, the specific patterns should be based on architectural usage, which involves *Video object detection* or other functionalities and preparation steps. In any case, GOBIA is designed to support various patterns in Phase II.

Phase III

Modifying Technology Selection Criteria. Selection criteria as listed in Table 4.13 can be modified or amended to accommodate future changes in architectural usage and selection patterns. These might lead to changes in the use of existing technologies and technology classes, but also to new technology classes.

For instance, when adding a hypothetical technology class *Multi-model Databases* to the data storage layer as discussed in the previous sub-section, the existing selection criteria must either be expanded to include *Multi-model Databases* (i. e., that they are selected for semi-structured data), or the criteria need to be changed to accommodate this new class.

Modifying the Tool Emission Process. The tool emission process when activating a technology class through a selection can be amended with capability-based filters that use an ABB or Phase I element type to provide necessary filter criteria. For instance, the selection of analytics tools is currently filtered by the analytical capabilities as needed by the underlying BI functionality.

5 Evaluation

In this chapter, the GOBIA method is applied to three additional use cases to conduct an evaluation. The FROG AIR case example has been designated as initial proof-of-concept in the *demonstration* phase in the DSRM, as it indicates that the GOBIA method is capable to address its objectives and requirements outlined in Section 3.7.2 (cf. [PTRC07, p. 55]). The cases presented here aim to solidify this demonstration and provide broader a illustrations of the GOBIA method and contribute to the *evaluation* phase of DSRM. First, the applied evaluation method is outlined in Section 5.1. Next, the GOBIA method is applied to three cases in Section 5.2. Each case description is followed by an execution of the three phases of GOBIA.DEV. Finally, the discussion of the evaluation results is conducted in Section 5.3. Here, the results of the case applications and their contribution for demonstrating the utility of the GOBIA method and fulfilling its solution requirements (cf. Section 3.7.2) are reflected upon.

5.1 Evaluation Method

Following the research method outlined in Section 1.2, the general goal of this evaluation is to systematically capture the results of an application of the GOBIA method and compare it to an expected baseline. An evaluation helps to observe the efficiency or effectivity of the artifact under consideration [PTRC07, p. 54].

However, several methods to evaluate design science research artifacts are available (cf. [HMPR04, p. 86], [PRTV12, pp. 401f]). These include observational methods such as case studies, analytical methods such as structural or dynamic analyses, experiments, software testing, and descriptive meth-

ods such as logical arguments and illustrative scenarios [HMPR04, p. 86]) [PRTV12, pp. 401f].

A study by Peffers et al. finds that the evaluation of *methods* primarily builds on *technical experiments*, *case studies*, and *illustrative scenarios* [PRTV12, p. 403].

Technical experiments evaluate the *technical performance* of a method using an "algorithm implementation" any artificial, real, or no data. These experiments are conducted in a synthetic environment, which allow for generalizability, but inhibit inferences to real-world usage [PRTV12, p. 401; p. 408]. However, a controlled environment allows to focus the study to selective qualities such as usability and utility, while a simulation with artificial data can add empirical insight [HMPR04, p. 86]. *Case studies* have an inherent observational element [HMPR04, p. 86]. They evaluate a method by applying it in a real-world situation and afterwards observing and studying its effect on said situation [PRTV12, p. 402]. *Illustrative scenarios* are a descriptive method [HMPR04, p. 86]. Scenarios focus on an application of a method to either a real-world or a synthetic, i. e., fictitious, situation. The main goal of this approach is to illustrate the "suitability or utility" of the artifact [PRTV12, p. 402]. An analysis by Prat et al. shows that descriptive approaches based on logical reasoning and analysis, such as illustrative scenarios, are dominantly used for artifact evaluation in the Information Systems domain in general [PCWA15, p. 249]. More specifically, according to their analysis, it is the most common method to evaluate usefulness [PCWA15, p. 253]. They further conclude that illustrative scenarios are commonly used to evaluate the "goal attainment" (in this context: fulfillment of solution requirements) and the "efficacy" of methods [PCWA15, p. 256].

The method of *logical arguments* [PRTV12, p. 402] uses information from a knowledge base, which can be built by analyzing related work and other "relevant research" in the domain of the designed method. Based on that, convincing and logical arguments to support a method's utility can be built [HMPR04, p. 86]. The GOBIA method was constructed (cf. Chapter 4) based on relevant research and findings related to representative technologies (cf. Chapter 2) and architectures (cf. Chapter 3). The FROG AIR case is employed

as running example and fulfills the role of an *illustrative scenario*, which is synthetic, but rooted in and aligned to a real-world use case in context of MIDAS and FROG AIR (cf. Section 1.3). Notably, illustrative scenarios employ either an analysis of results or logical reasoning for evaluation [PCWA15, p. 254].

The further evaluation of GOBIA is conducted using three additional *illustrative scenarios*, termed case applications. Like the FROG AIR case, they are synthetic scenarios, but are formulated based on real-world applications, use cases, and data. They are selected to cover a broader range of BI systems for decision making, which complements and extends the initial demonstration using FROG AIR. Such broader coverage means that a more heterogeneous BI and Big Data landscape can be represented, which is one of the solution requirements of the GOBIA method (cf. Section 3.7.2). The broader coverage of parts of the method aims to give additional indications for the overall utility provided by the GOBIA method.

5.2 Selected Case Applications

For further evaluation and demonstration of the GOBIA method, three application scenarios are employed. These application scenarios are selected to represent typical use cases in practice and cover a variety of elements in the GOBIA method. The FROG AIR case used before as running example is a mix of traditional and advanced analytics in a semi-structured environment with Web 2.0 data. Notably, as with FROG AIR, the cases are fictitious, but rooted in actual practical applications.

Stream Analytics. In a hospital asset management case, derived from an actual university project, a use case for streaming data analysis is executed with the GOBIA method. This case represents a mix of structured resting data and streaming data, which is worked on with a mixture of novel stream processing tools and traditional relational storage.

Advanced Analytics. Based on housing price data from a data science competition, advanced and exploratory analytics are demonstrated. Here,

additional requirement constraints aid in the technology selection process for the target BI system, which is based on traditional storage technology, but employs advanced analytics techniques.

***Big* Data Analytics.** Lastly, a case for academic plagiarism checking involving extraordinarily large-volumed data shows how conflicting goals and requirements lead to graceful abort of the GOBIA process in search for a novel algorithm, which is not covered by the standard building blocks of the GOBIA method.

The cases are structured similarly to the FROG AIR case demonstration throughout this work. However, all information is contained in one section and not spread among several sections. First, the case background and application scenarios are described. Based on that, the initial BI goals and requirements are derived. These are used to execute the three phases of GOBIA.DEV. For each phase, a description of conducted activities according to the GOBIA.DEV process model is presented, where notable decisions are explained textually, supported by graphical illustrations of decisions and results.

5.2.1 Stream Analytics: Hospital Asset Management

Hospitals are large facilities and have to deal with a large corpus of moving assets. Generally, this includes both people (i. e., patients or employees) as well as medical equipment such as mobile MRI devices or regular movable inventory such as spare beds. Because these assets are usually non-stationery, it is paramount to keep track of the whereabouts and ensure that these are available to staff upon request. Additionally, assets not in use must be at a known location, so that staff can also find free ones.

The University of Münster conducted a student project seminar during the winter term of 2016/2017 in conjunction with the German hospital *HPL*. Its goal was to implement a prototypical application that allows to locate and manage assets digitally using beacons. Beacons are usually small battery-powered devices that use Bluetooth or other wireless communications to promote information when signaled by an antenna (e. g., a smartphone) in

the vicinity of a few meters or centimeters. Using fixed antennas in a room, a beacon can be located by associating it with the known location of the particular antenna that senses it. The implemented prototypical web application can track beacons attached to assets and offers initial functionalities including real-time tracking of assets and associating it to rooms or other parts of a building.

Such digital tracking in real-time opens possibilities for further uses such as geofencing. Here, special behavior can be defined in a certain geographic location that is automatically triggered when entering it (i. e., a digital "fence" is placed around the location). In a hospital's case, "no-go areas" can be specified, which can ultimately help to prevent an asset from leaving the facility. Naturally, this can be narrowed down further to, e. g., a specific corridor therein.

At the HPL hospital in North-Rhine Westphalia, asset management is a challenge. Especially medical equipment is often not returned to their original location and has to be tracked down manually (e.g., by asking its last users). Further, a certain number of such assets is considered lost due to this and has to be purchased again. This leads to HPL buying surplus equipment to compensate for these challenges. Thus, it is feasible to explore means to improve this situation.

Building upon the prototypical application and the situation at HPL, GOBIA is used to find suitable technologies to support a potential full roll-out of a beacon-based asset management system there. For this, first, the situation at HPL and the prototypical application is described in additional detail. Based on this, goals and requirements are defined to start the GOBIA.DEV process, whose execution is documented in the subsequent sections.

Prototype Background and Infrastructure Considerations

The HPL hospital has approximately 30,000 assets in general (not including patients), which are located on their premises. Their facilities span a total of approximately 600,000 square meters. In the context of this HPL use case, assets refer to medical devices or other non-living equipment only.

Nevertheless, the system to-be-designed does not exclude a later extension for tracking patients or employees.

The developed prototype should demonstrate the general feasibility to track the aforementioned assets using beacon technology. Here, beacons using Bluetooth Low Energy (BLE) technology were used. Technically, a beacon is a BLE transmitter, whose signal strength can be measured by BLE receivers (i. e., antennas) inside buildings. The prototype connected the antennas directly to a Raspberry Pi mini computer, which redirects the measured signal strengths to a server via WiFi. Using signal strengths, not just from one antenna, software on the server can determine the room, where the asset in question is located (so-called *fingerprinting* or triangulation). The server software can run on any machine that can access the transmitted location data via network. Notably, the prototype is mostly a self-developed solution, including the hardware setup. The beacons and the base hardware of the Raspberry Pi were externally purchased.

This backend of the prototype is connected to a web-based frontend, which allows access to basic statistics and management functions. The latter include locating an asset in a specific room and listing all assets inside a room.

Besides expanding the requirements by more analytic functions and dealing with an increased scale of data, the hardware infrastructure must also be planned accordingly for a full roll-out. While the GOBIA execution focuses on the BI part of the system, it also defines requirements that are relevant for the underlying infrastructure. There are several possibilities to implement it. In addition to a self-developed infrastructure as in the prototype, multiple commercial solutions are available. These offer not just beacons or antennas, but also additional hardware that allows to scale such systems to large facilities.

In theory, it is possible to simply scale up the current setup. What has been demonstrated for a few rooms needs to be replicated to other rooms. However, several technical challenges come up due to sheer size of such endeavor such as the widespread availability of WiFi and power outlets in hospitals. Nevertheless, innovative solutions can help to overcome present

challenges, e. g., Osram light-bulbs with built-in antennas [317], which form a seperate WiFi mesh network (e. g., as defined in IEEE 802.11s [IEE11]). With that, less connected WiFi antennas are needed, which might be even powered by an Ethernet cable using Power over Ethernet (PoE) (cf. IEEE 802.3af [IEE03] and 802.3at [IEE09]).

For the evaluation of GOBIA, the focus is on the BI system, which is another important component of such a project. It is assumed that an infrastructure, which can satisfy explicit or implicit requirements related to it, will be implemented alongside the BI system. Further implications and relations of the requirements defined there to an infrastructure project are discussed during the execution of GOBIA.DEV.

Goals and Requirements

The main goal of this case application is to design a BI system that provides an analysis of asset movements over time to allow a full traceability of assets. First, location information of assets should be made available to a front-end system (i. e., to a map-based Web application). Second, events of assets entering an leaving certain rooms or other must be derived from an incoming stream. Based on this, tracking of assets in rooms over time should be realized. Third, with this information being available, geofencing should be implemented.

G_1: The system shall acquire access to asset locations in real-time.

Description: The basis for the whole system is that locations of assets are readily available for analysis. Moreover, timing is crucial for location availability. This means that the acquiring, processing, and analyzing of locations should avoid unnecessary delays. Essentially, this calls for asset locations being available to the system in real-time. Asset locations being available to the system does not imply that the locations must be persistently stored in a storage system at all cost.

G_2: The system shall provide analyses of asset locations over time.

Dependency: G_1
Description: Another system goal is to enable a time-series analyses based on the movement of the assets on-site from its origin to a destination. With these, hot spots for certain assets should be outlined and paths for single assets be documented.

G_3: The system shall provide geofencing for assets.

Dependency: G_1
Description: Certain assets should never leave associated rooms, floors, or buildings. By allowing to define digital boundaries (so-called geofencing, i. e., a digital location-based "fence"), for assets, users can be alerted if it is detected that an asset leaves its assigned boundaries. For this, naturally, the goal G_1 to provide timely location data on assets, must be fulfilled first.

After defining the goals, a scenario is used to illustrate these goals and provide additional details, which are not yet covered by the case description above. The primary target group of the application are specialists, who are responsible for managing the assets in the hospital. Not only do they need to know the current asset location, but they also need to be able to set and update geofencing boundaries. In particular, they use the analytics functionality to identify asset hot spots. These patterns can lead to business adjustments for asset management and procurement. For instance, if assets are often used at a particular location, it may be feasible to move the default storage for the asset to that location. Alternatively, another asset of the same type may be bought to address a particular station's needs.

However, more specific goals for time-series analyses need to be outlined. These sub-goals of G_2 need to address which goals the different types of analyses have with respect to the system and the stakeholder intentions (cf. Section 2.5.1).

Firstly, the system should be able to produce a heat-map or a comparable form of display, which needs to outline which asset is mostly found in which locations at specified points in time during the day and which locations

Figure 5.1: Example for a heat-map laid upon on a floor plan. Often visited areas such as corridor intersections (e. g., the T-junction on top left) appear thicker and are colored in yellow or even red as several data points converge on the same coordinates. Less frequented corridors have fewer data points on top of each other to visualize. Their visualizations are faint and therefore colored in green or even blue (e. g., the corridor between the top two T-junctions). Source: [404].

are "hot spots" for assets in general. The time frame to determine hot spots should be *14 days* (i. e. two weeks). All actual events (i. e. an asset entering or leaving a room) within that time are counted. Here, a room is a hot spot if it belongs to the top 30 %-quantile of all rooms with regards to the aforementioned events. Other variants to determine hot spots include a fixed threshold (all data points above the threshold are "hot") and other values for the top-quantile. A heat-map is a typical visualization for hot spots. Just as with heat, i. e. temperature, "hot" data is marked on a map or floor plan with red as "hot" color, which gradually fades to blue as "cold" color (cf. Figure 5.1).

Secondly, the system should provide average use duration for assets. During this, it should also be determined if assets are returned to their original location. In line with this analysis, the average time where a specific asset and all assets in general remain in a named location, should be calculated by the system. This leads to the following sub-goals, which all need to be satisfied by corresponding requirement so that their parent goal G_2 is fulfilled as well:

G_2: The system shall provide analyses of asset locations over time.

- $G_{2.1}$: The system shall provide hot spots by room and corridors over all assets.
- $G_{2.2}$: The system shall provide hot spots by asset for respective rooms and corridors.
- $G_{2.3}$: The system shall provide the average usage duration of an asset.
- $G_{2.4}$: The system shall determine whether an asset has been returned to its designated origin.
- $G_{2.5}$: The system shall provide the average visit duration for a specific and all assets by room.

Based on these goals, initial requirements are formulated. These include several functional requirements and one non-functional requirement.

Functional Requirements

R_1: The system shall provide an end-point to capture location events from the beacon system.

Description: Beacons physically installed on the assets can be captured by fixed antennas that capture their signals and measure the distance to the beacon. These antennas are installed throughout the hospital facilities so that beacons and, thus, assets can be tracked facility-wide. These antennas then send a signal that a beacon is near-by to a to-be-defined endpoint via network. This endpoint can be a simple URI that

ingests the event into a processing pipeline. From there, the event can be processes further. Naturally, this endpoint needs to be accessible by all antennas at any time without any delays due to parallel usage.
Requires: -
Relates to: G_1

R_2: The system shall interpolate asset locations from multiple beacons where necessary.

Description: An antenna may see beacons, which are outside its allocated room (e.g., in a corridor or in an adjacent room). Thus, a triangulation using two or more near-by antennas is optimal to determine an asset location with high accuracy[1]. Conclusively, it may be necessary to work through more than one location event from a beacon to determine an asset location. In addition to this, movement of assets can create several events in a short time frame for the same antenna and the same beacon. As it is not necessary to update the asset location more than once per second, unnecessary events may be discarded or used to improve accuracy of the location measurement.
Requires: R_1
Relates to: G_1

R_3: The system shall make the location data of assets available to the system.

Description: After assets have been mapped not just to coordinates inside the facility but also to names rooms, corridors, and buildings, this information must be made available to the analytics systems beyond. This includes that all necessary information about asset locations has been brought together and consolidated, previously.
Requires: R_1, R_2
Relates to: G_1

[1] Triangulation exploits the fact that the position of an asset can be determined if the distance to at least two measuring antennas is known.

R_4: The system shall have location data of named, actual locations inside the hospital facilities (buildings, rooms, and such) to map their location.

Description: The system needs to know in which physical locations the antennas reside to determine where an asset is absolutely located. While the distances to near-by antennas specify a relative position to these antennas, an absolute location is necessary for humans to find located assets (e. g., "Room A15" or "West Corridor, 2nd Floor"). These named locations have to be assigned to the antennas and must be known to the to-be-designed BI system. As the hospital has multiple buildings, where room numbers could be duplicated, at least the building has to be stored alongside a room or corridor name. It is sufficient, if named locations and associated antennas can be maintained by a systems expert, as physical locations rarely change after they have been initially set up in the system.
Requires: -
Relates to: G_1

R_5: The system shall have a database of all known assets, which is user-maintainable.

Description: All assets that should be tracked by the system need to be stored permanently, e. g., in a database. Such a database contains an identifier for each beacon, e. g., a hardware ID, and a display name that should be used for display purposes (e. g., "Mobile MRI - Building 42"). Further, the supposed location of an asset (e. g., "Room 1408") should be stored there, alongside the information if this asset has to be returned to that location after using it. The system should allow an authorized user of the system to register new assets and edit/remove existing ones.
Requires: -
Relates to: G_1

R_6: The system shall outline hot spots by room for all assets.

Description: The system shall provide a view, which outlines which rooms are hot spots that are most "frequented" by assets. A visit can

either be temporary while moving or a longer one, i. e., for permanent storage. A hot spot is in the top 30 % quantile regarding the number of visits in comparison to other rooms as outlined above. Hot spots are denoted from data of the last 14 days. For this requirement, it is necessary that asset locations are provided and rooms and assets are known. This means that goal G_1 must be fulfilled. For this, all requirements related to G_1 need to be satisfied.
Requires: G_1
Relates to: G_2

R_7: The system shall outline hot spots by room for specific assets.

Description: Similar to R_6, the system should outline asset hot spots, but in this case for specific assets, which the user can select. The system should outline the most frequently visited locations in the hospital for this asset. Hot spots are determined from data of the last 14 days. This requirement also relies on data dependent the fulfillment of goal G_1.
Requires: G_1
Relates to: G_2

R_8: The system shall provide average usage durations by asset.

Description: For a specific asset, the system should calculate how long it is used outside its designated origin until it is returned there. This requirement also relies on data dependent the fulfillment of goal G_1.
Requires: G_1
Relates to: G_2

R_9: The system shall indicate if an asset has been returned to its designated origin.

Description: If an asset has a designated origin, i. e., a room or place it should be returned to after usage, the system shall indicate visually if the asset has yet been returned there. Additionally, the time spent outside of its origin location should be displayed alongside it. With this, the decision can be made, whether the asset should be tracked

down or further action to initiate the return of the asset should be conducted. This requirement also relies on data, which is dependent on the fulfillment of goal G_1.
Requires: G_1
Relates to: G_2

R_{10}: The system shall provide average asset turnaround times for a specific asset by room.

Description: For a selected asset, the average duration an asset stays in a room should be calculated. This requirement also relies on data dependent the fulfillment of goal G_1.
Requires: G_1
Relates to: G_2

R_{11}: The system shall provide average asset turnaround times for all assets by room.

Description: For a selected room, the average duration an asset stays in there should be calculated. This requirement also relies on data dependent the fulfillment of goal G_1.
Requires: G_1
Relates to: G_2

R_{12}: The system shall allow users to define geofences for each asset.

Description: For each asset, a user shall be able to define geo-boundaries. These refer to rooms or to custom geological coordinates.
Requires: G_1
Relates to: G_3

R_{13}: The system shall alert users when geofences are violated.

Description: If an asset leaves its defined boundaries (R_{12}), an alert is shown to users, which notifies them, that an asset has left its boundaries and shows where its current or last known location is.
Requires: G_1
Relates to: G_3

R_{14}: The system shall be a user-accessible web application.

Description: A Web-interface should be the system's graphical user interface. All interactions with the BI system take place here.
Requires: -
Relates to: G_{2-3}

Non-Functional Requirements

R_{15}: The response time (analysis latency) for location events should be below 30 seconds.

Description: To enable geofencing, the information about an asset's updated location must also be made available quickly, i. e., in "real-time" (cf. Section 2.2.6). As soon as an asset leaves its allocated boundaries, appropriate measures must be taken. For that, a timely reaction is obligatory.
Relates to: G_1, R_3

This use case contains BI-related requirements, but also web-based accessibility as well as data processing as functional requirements, of which the latter are used besides the BI functionality in front-end applications. As the goal of HPL is to have one system for all of those, all requirements must be adhered to when choosing the proper technology. The non-functional requirement to consider is a latency requirement: The data must be available in real-time (i. e., within seconds-time). This is due to geofencing, which necessitates an immediate availability of new location data to determine if an asset is outside of its allowed bounds. This is especially important if other actions are depending on this, e. g., an alarm going off in case of an asset being moved.

Phase I

In total, there are 7 goals and 15 requirements for the HPL case (cf. Table 5.1). These are the initial inputs for GOBIA.DEV Phase I and are assumed to be exhaustive, i. e., they cover the contents of the HPL case completely.

Table 5.1: Initial input data items from the HPL case to start the GOBIA.-DEV process.

Input	Artifacts from the HPL case
Initial Goals	$G_1, G_{2.1-2.5}, G_3$
BI Requirements	R_1 – R_{15}

Table 5.2: HPL case: Separation of input data into GOBIA.DEV Phase I into functional and non-functional requirements as first step.

Requirement Type	Requirements from the HPL case
Functional Requirements	R_1 – R_{14}
Non-Functional Requirements	R_{15}

Extract Functional and Non-Functional Requirements. This distinction was made above, thus the extraction is directly derived from there and is listed in Table 5.2.

BI functionality and BI data entity definition. R_1 denotes the capture end-point for the *raw beacon data*. R_3 combines these to *assets location events*, for which R_2 states how to attain them using the *raw beacon data*. The two aforementioned pieces of data are mapped onto two respective BIDEs. All this data is considered to be streaming data due to its high-velocity. It is assumed that these are semi-structured data objects carrying event information. R_6 and others demand that these must be permanently stored a part of the *asset location history*. Thus, R_2 is set aside for the next step, where data preparation is needed. R_4 defines additional metadata such as names for the physical locations so that named rooms, corridors and building can be identified. R_5 leads to a list of named assets, which are traceable by the system. R_{12} refers to *asset geofences* as optional data for aforementioned assets. Both denote structured *asset master data*, which is created as joint BIDE.

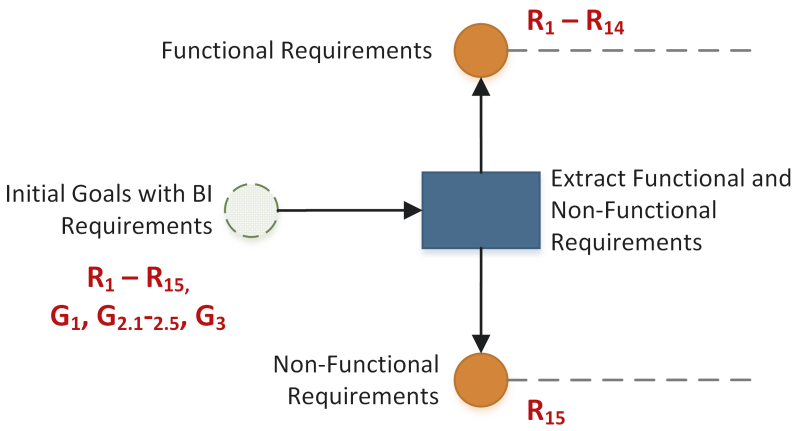

Figure 5.2: First part of execution of GOBIA.DEV Phase I in the HPL case.

BI functionalities are derived from the requirements with the help of the mapping table in Table 4.5. R_6 and R_7 require that hot spots are detected. Essentially, this requires a *ranking* of asset visits per room over the last 14 days. As this is a function on aggregated data, R_6 and R_7 are covered by a joint BI functionality of category *Aggregated Measures & Descriptions* (BF_1). Similarly, R_{10} and R_{11} rely on an average calculation on turnaround times, which can be covered by a joint *average* BI functionality from the same category (BF_2). Notably, a turnaround time also involves arithmetic calculations on data. Lastly, R_8 denotes average usage times of an asset outside its origin location, which results in another aggregate measure (BF_3).

R_{13} specified the user alert functionality. As this is neither an aggregate measurement nor a descriptive or predictive analytics functionality, it leads to a BI functionality from the *Direct Information Access* category, which will rely on specific data data preparation in the next step (BF_4). R_{14} is set aside, because it mandates a web-based user-interface as means of accessing the provided BI functionality and therefore does not directly relate to BI functionality or data entities. Lastly, R_9 needs a combination of asset location

Table 5.3: BI functionalities and BIDEs for the HPL case derived in GOBIA.-DEV Phase I.

Element Type	Identifier	Name	Element
BI functionality	BF_1	**Frequency ranking** of asset visits	Aggregate Measures & Descriptions
	BF_2	Calculate **average** turnaround times by room	Aggregate Measures & Descriptions
	BF_3	Calculate **average** usage times by asset	Aggregate Measures & Descriptions
	BF_4	**Provide** geofencing violation alert	Direct Information Access
	BF_5	**Provide** asset return indicator	Direct Information Access
BI data entity	$BIDE_1$	Asset location events	*Semi-structured* Streaming BIDE
	$BIDE_2$	Asset master data	*Structured* BIDE
	$BIDE_3$	Asset location history	*Structured* BIDE

and asset master data to determine if an asset has been return to its origin location yet (BF_5).

The created elements are summarized in Table 5.3. The mapping process is visualized in Figure 5.3.

Check alignment between BI functionality and BIDE. R_2, which was previously set aside, provides the connection between raw beacon update events and the triangulated asset location events. However, this is only considered when "unfolding" the BIDE into data sources in subsequent phases. Nevertheless, the connection between BF_4 and $BIDE_1$ as well $BIDE_2$ needs data preparation of the event stream $BIDE_1$ by *filtering* location updates, which violate the geofences specified in $BIDE_2$. Thus, DP_1 is added (cf. Table 5.4). As BF_5 is a functionality on user request and

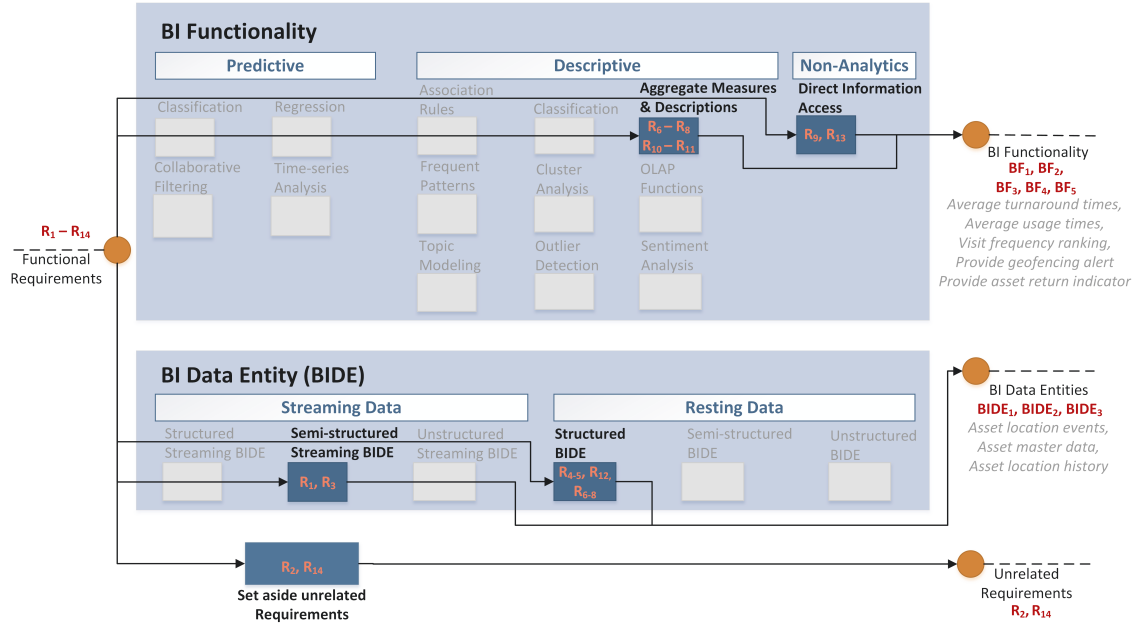

Figure 5.3: Second part of execution of GOBIA.DEV Phase I in the HPL case: Selection of BI functionality and BI data entities.

Table 5.4: Data preparation steps added for the HPL case in the course of GOBIA.DEV Phase I.

Element type	Identifier	Name	Inputs	Outputs
Data preparation	DP_1	Filter geofencing violation events	$BIDE_{1-2}$	BF_4

no active push, it accesses location history and master data, but does not need a dedicated combination activity. There are no further preparations needed between the other BI functionalities. BF_{1-3} can be calculated by using location history data alone ($BIDE_3$), whereas BF_5 needs origin info from the master data as well as current location data ($BIDE_3$ and $BIDE_2$). Notably, this is decided based on calculations only. For a display in a web application, additional asset and room data is likely to be shown alongside this data. BF_4 needs a combination of current events conjointly with master data ($BIDE_1$ and $BIDE_2$)

Check elements alignment to requirements. The single non-functional requirement in this case does not necessitate further alignment. The performance requirement R_{15} is covered by acknowledging the streaming source for location event updates. The previously set-aside requirement R_{14} — as in the FROG AIR case — specifies that the resulting application should have a user-accessible web front-end, which is not inhibited by any of the defined functionalities. These requirements become more relevant in the course of the next phases and in the following implementation after the GOBIA method has concluded. Likewise, the set-aside requirement R_2 is not relevant at this point yet. The other elements, which have been created, are aligned to the requirements.

Compose Architecture. The final abstract BI architecture is depicted visually in Figure 5.4. It illustrates the connections between the BI functionality, data preparation steps, and BIDEs.

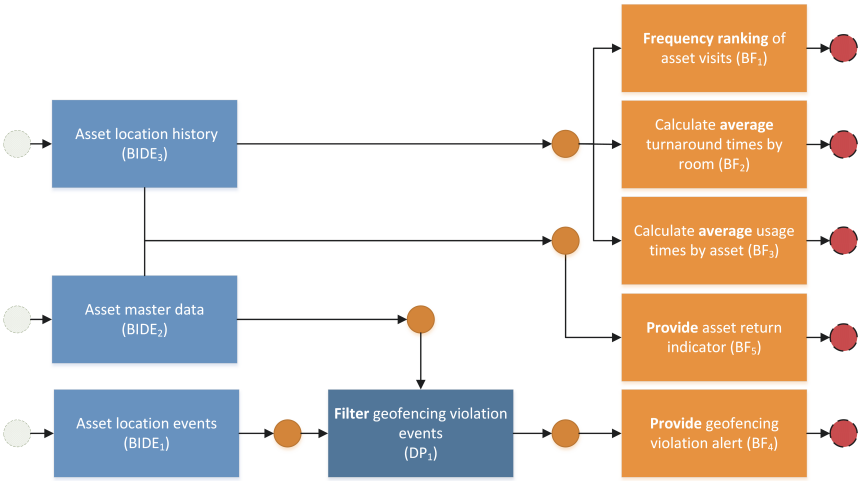

Figure 5.4: GOBIA.DEV for the HPL case: Abstract BI architecture of Phase I depicted using a Petri net with the three element types BIDE, data preparation, and BI functionality ordered from left to right.

Phase II

Separate by BI functionality and subordinate data preparation and BI data entity elements. The abstract BI architecture in Figure 5.4 also pictures the three element types separated by color from left to right. Now, for each of these elements, appropriate ABBs need to be defined. In particular, the processing steps to achieve the BI functionality and to prepare the data need to be detailed more closely.

Add Analytic/Processing/Storage/Generation ABBs. For each target BI functionality, denoted on the right side of Figure 5.4, a chain of building blocks leading up to them, is created and denoted as tactical plan. These five tactical plans denote the flow of data through the architecture to attain the target functionality. For example, BF_1 for average usage time calculation

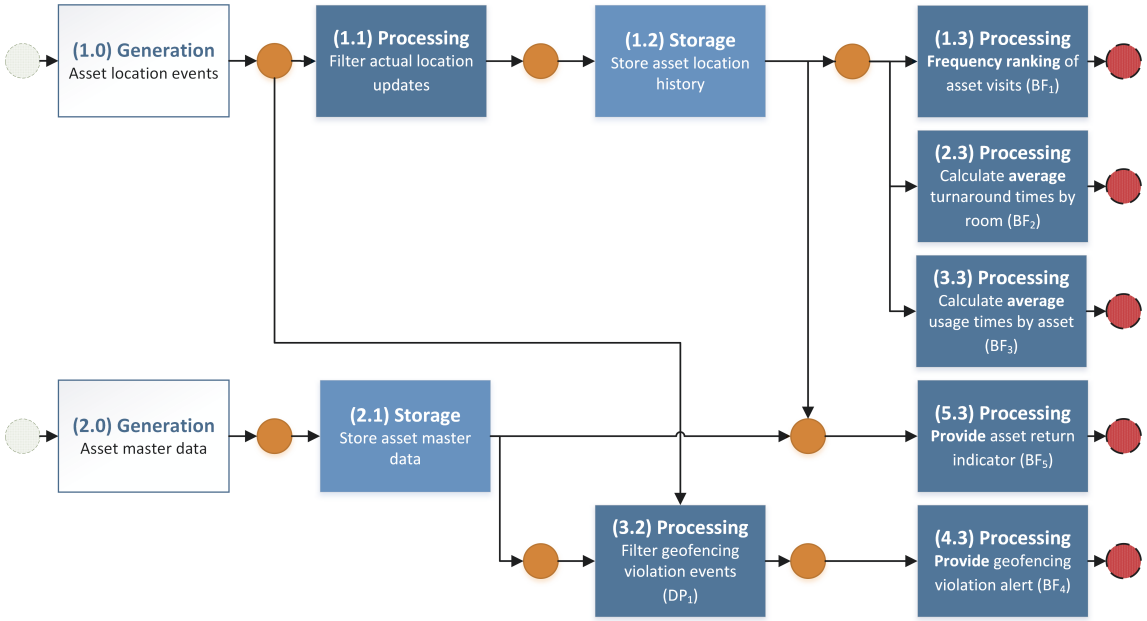

Figure 5.5: GOBIA.DEV Phase II for the HPL case: Tactical plan perspective of the semi-concrete BI architecture, which is the main output of Phase II. A unified view on tactical plans outlines value generation using ABBs.

relies on historized asset location data, where as BF_4 relies on a preparation of a data stream and lookup in the asset master data.

Data generation ABBs are added for BIDE, which is generated and fed from outside into the BI system. Here, $BIDE_1$ and $BIDE_2$ are mapped to generation blocks. However, $BIDE_3$ is data that is derived from $BIDE_1$ in the scope of the BI system. Thus, it is mapped to a storage building block only. Conceptually, data is diverted from the event steam $BIDE_1$ and stored. The stream of asset location events is based on triangulation of beacon signals and a mapping to rooms and assets. Although this can be implemented manually (e. g., by using a stream windowing function for triangulation and a mapping onto known assets), state of the art hardware solutions for beacons and their receiver nodes already include triangulation functionality (e. g., cf. [240]). Such systems also cover R_2 and parts of R_4, although the asset database is in the present BI system's realm. Thus, it is assumed that location events are streamed into the system according to the beacon update frequency, which is discussed in the subsequent phase. In addition, master data is to be stored permanently and needs a storage ABB as well. The filtering of location events (DP_1) represents a typical data processing activity and is mapped to a processing ABB. It should be noted that no storage building blocks are added here to accommodate the data source properties as streaming data, where no unnecessary latencies should be introduced to allow for fast alerting. Lastly, the mapping of BI functionality must be decided upon. Frequency ranking (BF_1), averages over aggregate data (BF_{2-3}), and non-analytics functionality do not represent advanced analytics or complex OLAP functionality. Thus, they are technically processing activities and mapped to corresponding processing ABBs.

Compile into Architecture grouped by Analytics ABBs. Figure 5.5 pictures all five tactical plans (cf. Table 5.5) in combination. A separate view per functionality and tactical plan is omitted at this point, but can be derived from the combined view. Figure B.6 in Appendix B depicts a full model as semi-concrete BI architecture, which builds on the aforementioned building blocks, structured into architectural layers. An overview of building blocks is listed in Table 5.6.

Table 5.5: GOBIA.DEV Phase II: Tactical plans for HPL.

Identification	Target BI functionality
TP_1	Frequency ranking of asset visits (BF_1)
TP_2	Calculate average turnaround times by room (BF_2)
TP_3	Calculate average usage times by asset (BF_3)
TP_4	Provide geofencing violation alert (BF_4)
TP_5	Provide asset return indicator (BF_5)

Table 5.6: GOBIA.DEV Phase II: List of Architecture Building Blocks through all HPL tactical plans.

ABB	Type	Description	Tactical plans
1.0	Generation	Asset location events	TP_5
2.0	Generation	Asset master data	TP_{4-5}
1.1	Processing	Filter actual location updates	TP_{1-4}
2.1	Storage	Store asset master data	TP_{4-5}
1.2	Storage	Store asset location history	TP_{1-4}
3.2	Processing	Filter geofencing violation events (DP_1)	TP_5
1.3	Processing	Frequency ranking of asset visits (BF_1)	TP_1
2.3	Processing	Calculate average turnaround times by room (BF_2)	TP_2
3.3	Processing	Calculate average usage times by asset (BF_3)	TP_3
4.3	Processing	Provide geofencing violation alert (BF_4)	TP_4
5.3	Processing	Provide asset return indicator (BF_5)	TP_5

Phase III

Decompose into Architecture Building Blocks and Assign Layers. After a decomposition of the semi-concrete BI architecture into single ABBs for technology selection, the mapping into layers of the GOBIA.REF technological reference architecture takes place. As all functionalities indirectly (BF_{1-4}) or directly (BF_5) depend on ABB 1.0 *Asset location events*, layer assignment starts with the data.

Thus, ABB 1.0 is mapped to both the data generation and the data acquisition layer. Streaming data requires technical management to be handled efficiently, which appropriate acquisition tools can provide. For the *Asset master data* (2.0) it is assumed that a copy of asset information is pulled from an operational database. Therefore, both generation and acquisition layer are selected.

The filtering of actual location updates (1.1) out of repeatedly streamed location information is mapped to a processing layer, as it works directly on streaming data. Similarly, 3.2 to filter geofencing violations are mapped to the processing layer for the same reason as well. Storage ABBs 1.2 and 2.1 are assigned to the storage layer to select storage technologies. Lastly, the main BI functionalities of the five tactical plans BF_{1-5} are also assigned to the processing layer to find appropriate processing technologies.

Select Technology Classes. The selection process is depicted sequentially from data source to data processing blocks.

> **Data Acquisition Technology.** Incoming *asset location events* (1.0) are categorized as semi-structured *streaming data* by the data generation layer. Thus, *Stream Processing Engines* and *Streaming Data Collection Tools* are selected to handle incoming streaming data. If data flow functionality with queuing is needed will be decided after GOBIA.DEV when specific loads are determined and empirically tested. The corresponding tools listed in Section C.1.1 are emitted, which are *Apache Kafka*, *Apache Flume*, *Apache NiFi*, *Apache Spark (Spark Streaming)*, *Apache Storm*, *Apache Flink*, and *Apache Kafka (Kafka Streams)*. In contrast to that, *asset master data* (2.0) is regular resting data, which is ingested

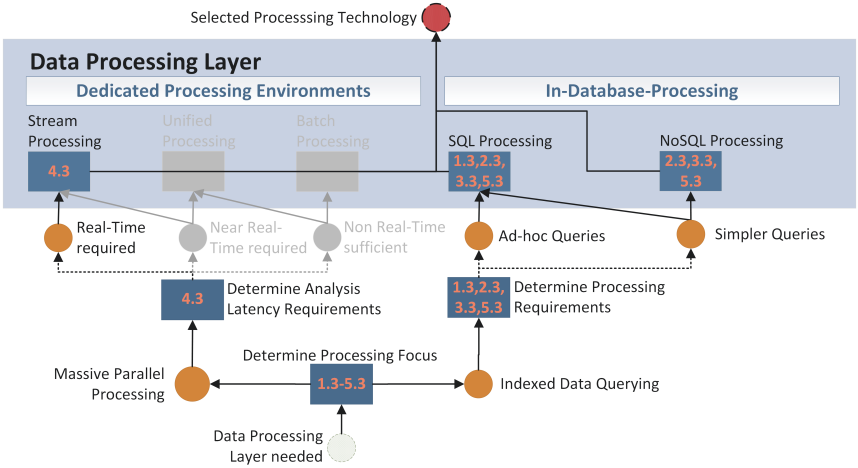

Figure 5.6: GOBIA.DEV Phase III for the HPL case: Illustration of the selection of processing technology in Phase III.

into the BI system (cf. R_3 and R_4). For resting data, *Data Integration and ETL Tools* are activated. For this, the tools in the repository are *Pentaho Data Integration (Kettle)*, *Talend Open Studio*, and *Apache Sqoop*.

Data Storage and (Pre-)Processing Technology. The incoming and acquired master data should be stored (ABB 2.1). To select a storage technology, data volume must be estimated first to decide whether distributed storage is necessary. There are approximately 30,000 assets (see above), whose master data is structured, textual data. Assuming a size of 100 KB for each record per asset, overall data size is approximately 3 GBs. Even a ten-fold increase in data size per record or of asset numbers would easily fit into a centralized storage system. As the assessment of data variety has already concluded that asset master data is structured, *Centralized SQL RDBMSs* are selected and *MySQL* and *PostgreSQL* are emitted from the tool repository.

Before storage of the asset location history is discussed, technology for processing ABB 1.1 for filtering actual location updates is selected. As asset location events are sent constantly, even if the actual room has not changed, changes in locations have to be detected and directed to permanent storage. As high-velocity of streaming data is made available, the focus of data processing is on MPP, also because the filtering activity does not rely on indexed data querying capabilities. The requirements on analysis latency can be derived from R_{15}, which prescribes an analysis latency below 30 seconds to react to location updates. 30 seconds fall exactly between second and minute spans (cf. Section 2.2.6), which discriminate between near real-time and real-time requirements. However, as this filtering activity only leads to storage and analyses, which do not necessitate an immediate reaction, *near real-time* is deemed sufficient. This decision leads to *Unified Processing* and *Stream Processing* systems. The GOBIA tool repository emits *Apache Spark*, *Apache Storm*, *Apache Flink*, and *Apache Kafka (Kafka Streams)*. The filtered asset location change events need to be stored next (ABB 1.2). For that, the data size of the location history needs to be estimated. The overall volume depends not only on the 30,000 assets, but also on how often they change and how long the history should be retained. With respect to the latter, several of the four final BI functionalities related to historical location access only need recent data. The calculation of hotspots (1.3) is specified for data of the last 14 days. A return indicator (4.3) relies on the most recent location data only. However, no timespan is specified for average turnaround time by room (2.3) and average usage time calculation by asset (3.3). Thus, it is assumed that history needs to be available for the system's life cycle. A location history record consists of a room and asset identifier and start and end date of a "visit" in a room. Cautiously assuming that each of those can be represented as 32-bit Integer value and adding metadata overheads, it is assumed that each entry is $\tilde{1}00$ Bytes in size. Even if every one of the 30,000 assets were to change its location every 15 minutes on average, i. e., approximately 100 times a day, only 10 KB per asset per day is generated. This accumulates

to approximately 300 MB per day for all assets, and 100 GB per year. Assuming a life cycle of the system of 15 years, 1.5 TB of data still fits into a centralized storage system. This holds true even if the number of assets over time should double or triple. As the data structure of these visits is fixed, a *structured* storage is selected, which leads to *Centralized SQL RDBMS*, where again *MySQL* and *PostgreSQL* are emitted from the repository.

The last preprocessing activity is the filtering of geofencing violations (3.2). While this processing block accesses the asset master data to get the latest geofencing boundaries, its main focus is on processing the high-velocity event stream. Access to master data is only relevant from a compatibility point-of-view. As before, the processing focus of boundary checking is on *MPP* instead of queries on indexed data. Especially in this case, *real-time* analysis latencies are required to allow for quick geofencing boundary violation detection (R_9). This leads to the selection of *Stream Processing*, represented by *Apache Storm*, *Apache Flink*, and *Apache Kafka (Kafka Streams)*.

Data Processing Technology. Lastly, technologies for the BI functionalities BF_{1-5} represented by ABBs $1.3 - 5.3$ are selected. Processing blocks 1.3, 2.3, and 3.3 similarly rely on the asset location history to calculate various aggregation queries. Instead of focusing on MPP, they require *Indexed Data Querying*. 1.3 needs to count and rank data for a certain subset. This combination makes it a more complex querying functionality. Thus, *Ad-hoc Queries* are required, which can be satisfied by *SQL Processing*. 2.3 and 3.3 rely on simple arithmetics (turnaround time) and average aggregate calculations, which are *Simpler Queries* and can be satisfied by both *NoSQL Processing* and *SQL Processing*. For SQL processing, RDBMSs *MySQL* and *PostgreSQL* are emitted, whereas the selected tools for NoSQL processing are *MongoDB*, *Couchbase*, *Redis*, *Riak KV*, *Apache Cassandra*, *Apache HBase*, and *Neo4j*. Next, processing block 5.3 is similarly a simpler query on indexed data, which compares the home location of an asset with its current location. Conclusively, the previous selection is repeated here. Lastly, 4.3 relays geofencing alerts

Table 5.7: Technology classes selected for HPL in GOBIA.DEV Phase III.

Tactical plan	Element	Layer	Technology Class Selection
TP_{1-5}	(1.0) Generation	Data Acquisition	Streaming Data Collection Tools Stream Processing Engines
TP_{1-5}	(1.1) Processing	Data Processing	Stream Processing Unified Processing
TP_{4-5}	(2.0) Generation	Data Acquisition	Data Integration and ETL Tools
TP_{4-5}	(2.1) Storage	Data Storage	Centralized RDBMSs
$TP_{1-3,5}$	(1.2) Storage	Data Storage	Centralized RDBMSs
TP_4	(3.2) Processing	Data Processing	Stream Processing
TP_1	(1.3) Processing	Data Processing	SQL Processing
TP_2	(2.3) Processing	Data Processing	SQL Processing NoSQL Processing
TP_3	(3.3) Processing	Data Processing	SQL Processing NoSQL Processing
TP_4	(4.3) Processing	Data Processing	Stream Processing
TP_5	(5.3) Processing	Data Processing	SQL Processing NoSQL Processing

to be shown to users. No additional processing to the data coming from the previous filter activity 3.2 need to be applied. Thus, the selection of 3.2 is repeated as the boundaries of previous processing activity are not left (cf. Section 4.3.2). The selection of processing technologies is illustrated in Figure 5.6.

An overview of all selections is presented in Table 5.7.

Create Compatible Technology Sets. After technology classes and tools have been chosen, compatibility is checked (cf. Section C.2). For the stream processing pipeline, the relevant information is whether processing capabil-

ities are available to disseminate actual location updates, but also whether RDBMS data can be ingested to use it during streaming as well as be a target for persisting filtered streaming data. This applies to Apache Kafka with Kafka Streams as well as partially to Apache Storm, where single tables can be loaded. As ingestion tools for master data, all three ETL tools are compatible to receive structured data and write it. Processing needs for $TP_{2,3,5}$ include both SQL but also NoSQL processing. However, as no NoSQL data stores are selected for storage, only SQL processing remains, which is compatible to both MySQL and PostgreSQL. Apache NiFi and Apache Flume can only be utilized, when used conjointly with Kafka, i. e., when feeding Kafka. Kafka is also compatible with Storm, but Storm can also be used alone.

For stream processing, the compatible combinations are as follows:

1. (Apache Kafka, (Apache Storm | Kafka Streams))
2. Apache Storm
3. *((Apache NiFi | Apache Flume), Apache Kafka, (Apache Storm | Kafka Streams))*

Options in (parentheses) denote a selection of one of the tools therein. Here, the last possible combination in *italics* is put aside, as Kafka offers stream data collection capabilities such as queuing and there is no apparent reason to duplicate this functionality using Flume or NiFi.

Thus, the tool combinations for this case are: **(Talend Open Studio, Pentaho Data Integration, Apache Sqoop)**, **(MySQL, PostgreSQL)**, **(Apache Storm, (Apache Kafka, (Apache Storm, Kafka Streams)))**.

Compose BI Architecture Alternatives. Lastly, the resulting tools from before are compiled into several BI architecture alternatives. Figure 5.7 illustrates a sample BI architecture alternative using Talend Open Studio, MySQL, and Apache Kafka with Kafka Streams. The other BI architecture alternatives are constructed analogously. Figure B.8 in Appendix B depicts a simplified combination of all alternatives using one shared Petri net.

After GOBIA.DEV concludes, these alternatives can be subject to further evaluation and be used in the continued planning of the BI system. This is

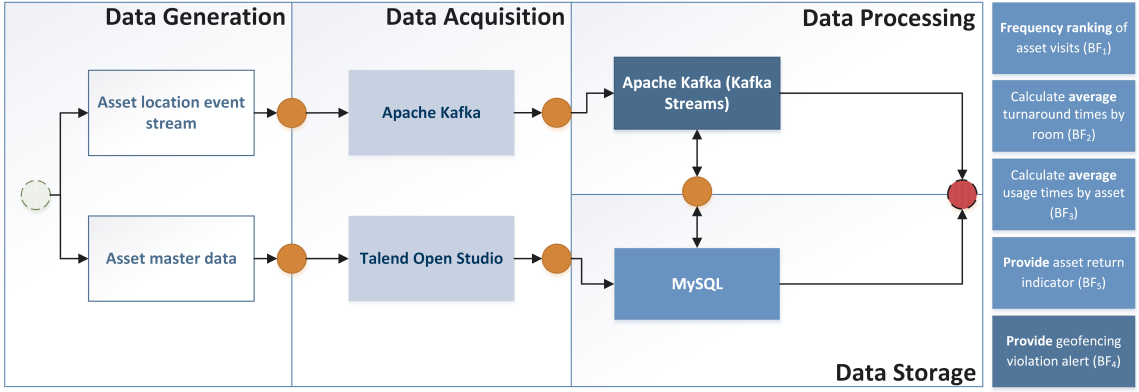

Figure 5.7: GOBIA.DEV Phase III for the HPL case: Illustration of a particular BI architecture alternative on all layers of Phase III. Five data uses (BF_{1-5}) are shown right to the model. Data flows are not depicted separately.

concluded with the selection of a final BI architecture, which is implemented. This also includes the web-based graphical interface as final part of the *data usage* phase of the analyzed data, which is not in the scope of the GOBIA method.

5.2.2 Advanced Analytics: Housing Price Prediction

Kaggle Incorporated's website `kaggle.com` [223] offers various "data science challenges", where companies can upload a data set and ask for creative solutions for machine learning problems. Here, the Kaggle case "Housing Prices" [222] is used to construct a scenario for an application of the GOBIA method. This housing price scenario is supplied by Kaggle to train data scientists [vdA14] in two areas: Feature selection and regression techniques. The goal of this challenge is to use the supplied data set with features of houses to predict their sales price. The data set [221] features 79 attributes that describe various contingencies of houses in the town of Armes, Iowa in the United States. It is based on a data set originally compiled by De Cock [De 11]. The explanatory attributes include, e. g., a zoning classification such as agriculture, residential, or commercial, the number of fireplaces in a home and their quality, or the location and size of the garage. The attributes are both categorical (e. g., the quality of a garage, if existent, is measured on five-point categorical scale between "excellent" and "poor") and numerical (e. g., lot size in square feet). Both a training and a test data set are supplied. The latter is used to build and "train" a predictive model. In that training data set, actually observed sales prices are listed to verify the prediction results and assess the prediction quality. Test data is used to test the finally built model. Notably, proper test data *should* not be used to tune the model, as it introduces a bias to this data, i. e., an overfitting to the predictive model. For that, a distinct validation data set can be built or should be supplied [124].

Such Kaggle challenges resemble cases of *exploratory analysis*, where the specific business question is not or only partially known, but data sources are known. Here, a rich data set is given and a basic desire to "predict prices" is expressed. However, to be suitable for the GOBIA method, a business question is needed to formulate specific requirements. This is achieved by

reformulating an open-questioned exploratory analysis case as dedicated business question (cf. Section 3.2). Here, the business question is to identify *features*, i. e., attributes, of houses, which can reliably predict their price. For instance, a real-estate agency may have collected or procured several features of their to-be-sold properties and is interested in quickly determining suitable market prices for both existing as well as new homes in their portfolio. To illustrate the formulation of exploratory analyses as business question, this context is employed. In this example, fictitious company **Kaggle Real-Estate (KRE)** has the desire to build a BI system for exploratory analysis of housing data, which should yield important house features that determine sales prices. Given is the Kaggle housing data set with approximately 1,500 data objects and 79 attributes and a training set to actually build the price predictors. In this case, the training set consists of 1,500 (different) house objects including sales prices.

Goals and Requirements

The overall goal of KRE it to provide a system that allows to ingest and analyze incoming housing data in CSV format and detect the most discriminate features, which are then used to predict sales prices for new sales in the real-estate portfolio. The housing data is divided into training data (i. e., houses with explanatory features from the portfolio, which are not yet sold) and test data (i. e., previous housing sales).

The initial **goals** of the exploratory BI system can thus be determined as follows:

G_1: The system shall acquire access to housing data.

Description: The data basis for the exploratory system is made available in machine-readable format at a central location.

G_2: The system shall detect important house features driving sales prices.

Dependency: G_1
Description: When 79 or more features of a house are known, it is desirable to determine which of those drive the sales price in a house's

region. This is not only useful for price prediction, but also for real-estate agents who can focus on marketing valuable characteristics of the real-estate portfolio. For instance, if fireplaces were an important factor for higher sales prices, agents could point out detailed features of a fireplace inside a house to customers to increase their willingness to pay more for a house.

G_3: The system shall provide predictions for housing prices in the portfolio.

Dependency: G_1
Description: After finding out the most important explanatory features, a predictive model should be built, which can be fed with properties of (old and new) real-estates in the portfolio and provides a price prediction for them.

An application *scenario* illustrates these goals and provides additional details. First, the primary target group of the system are data scientists, who apply various algorithms and methods for feature extraction and price prediction (i. e., regression). Although the data is in machine-readable format, the data scientists need to cope with anomalies and imperfections in the data, since these can influence analytics, e. g., missing values. Data preparation includes transforming data types, and performing normalization of values, and adjusting values (e. g., from categorical to numerical). Based on this, feature extraction and prediction are performed. However, the nature of exploratory analysis dictates that several methods and algorithms can be applied and their results compared. The outcome of feature extraction and analysis is used by the data scientists to educate executive staff and real-estate agents. Predicted values should be stored permanently for electronic retrieval later. Based on this scenario, no further goals need to be added. Furthermore, no sub-goals are necessary to capture the contingencies of this scenario.

Next, the initial **functional requirements** can be derived:

R_1: The system shall provide access to the training data.

Description: The system needs to be able to physically access the training data in CSV format. The training data is used to build and train

a predictive model as it provides both historic sales and explanatory features of real estate.
Requires: —
Relates to: G_1

R_2: The system shall provide access to the test data.

Description: The system needs to be able to physically access the test data in CSV format. The test data represents yet unknown data to the model. The built model is used for predicting these data items. Here, these are houses in the portfolio, which have not been sold yet.
Requires: —
Relates to: G_1

R_3: The system shall pre-process the data to enable meaningful analytics.

Description: Data preparation steps as outlined in the scenario above might be necessary to bring the data into a proper form to conduct various forms of analytics, i. e., feature extraction and prediction.
Requires: G_1
Relates to: G_2, G_3

R_4: The system shall identify the most relevant features for house sales prices.

Description: Through means of feature extraction techniques, those ones of the 79 input features should be identified, which have the most explanatory power.
Requires: G_1
Relates to: G_2

R_5: The system shall store the most relevant features for later retrieval.

Description: To make extracted features available after their calculation has concluded, they should be stored for later retrieval.
Requires: G_1
Relates to: G_2

R_6: The system shall provide a prediction of the housing prices.

Description: With help of the extracted feature, a predictive model should be built which can deliver predictions of high quality. The high quality of the predictions is ensured using the training data.
Requires: G_1
Relates to: G_3

R_7: The system shall store the predicted hosing prices for later retrieval.

Description: After predictions have been calculated they should be stored so that they can be retrieved later. Uses in new scope outside this BI system might include viewing the predictions or use them as basis to build an automated price prediction system at a later point in time.
Requires: G_1
Relates to: G_3

The functional requirements are completed by the following **non-functional requirement**:

R_8: The price predictions should be qualitative according to agreed upon measures and criteria.

Description: Although several measures to evaluate the predictive power of a model exist, there are several agreed upon measures and criteria to evaluate the prediction. For instance, the *Mean Square Error (MSE)* should often be minimized. The MSE measure deviations between predicted and actually observed values in the training set. However, no general thresholds apply, but the interpretation of the value rather depends on the present contingencies. For instance, if another method exhibits a smaller MSE but needs significantly more features for that, a trade-off between these optimizations target has to be chosen cf. [HKP11, p. 452].

Table 5.8: Initial input data items from the KRE case to start the GOBIA.-DEV process.

Input	Artifacts from the KRE case
Initial Goals	G_{1-3}
BI requirements	R_1 – R_8

Additionally, the following **constraint** is specified:

C_1: The analytic tools should be versatile and offer several methods for feature extraction and value prediction.

Description: In exploratory analyses several methods for analysis might be tried out, before finding the proper and final one to build the predictive model. Thus, this constraint should ensure that tools selected for implementation support more than one method for feature extraction and prediction of values (i. e., regression).
Attached to: R_4, R_5

As noted in Section 4.3.1, constraints are attached to the respective requirements and considered when checking the alignment to these throughout GOBIA.DEV and post-GOBIA.

Phase I

The three goals and eight requirements for **KRE case** are listed in Table 5.8.

Extract Functional and Non-Functional Requirements. The extraction of functional and non-functional requirements leaves only these elements for the next. As in the cases before, this distinction is already given by the requirement definition above (cf. Table 5.9). R_{1-7} remain as functional requirements and are used in the next step of GOBIA.DEV Phase I. $R_8's$ functionality is side-routed until alignment to requirements is checked later in this phase.

Table 5.9: KRE case: Separation of input data into GOBIA.DEV Phase I into functional and non-functional requirements as first step.

Requirement Type	Requirements from the KRE case
Functional Requirements	R_1 – R_7
Non-Functional Requirements	R_8

BI functionality and BI data entity definition.

BI functionalities. R_6 provides the main functionality of the KRE case. It is a prediction of prices, which is continuous data. Therefore, a *regression* BI functionality is appropriate and results in BF_1 (cf. Table 4.5).

BI data entities. R_1 and R_2 define the data that is subject to analytics. No other data is defined as input for the BI system. For R_1, *Real-estate sales (training)* is defined as $BIDE_1$, whereas R_2 leads to the second data entity: *Real-estate portfolio (test)* as $BIDE_2$. Both training and test data are, in spite of the CSV format, ultimately tabular data. Thus, working with a *structured BIDE* is appropriate. This might lead to necessary acquisition and preprocessing in later GOBIA.DEV phases. Apart from that, these fixed data sets unsurprisingly represent *resting data*.

Set aside requirements. R_3 and R_4 do not refer to BI functionality in the GOBIA.REF functional reference architecture. They might be considered for data preparation after functionality and data entities have been defined. R_5 and R_7 require a specific form of persistence on data that is generated in the process of analyzing the input data. Such persistence blocks are only decided upon in the next phase and are thus set aside.

The resulting elements are listed in Table 5.10, while the definition process with respect to the GOBIA.REF functional reference architecture is illustrated in Figure 5.8.

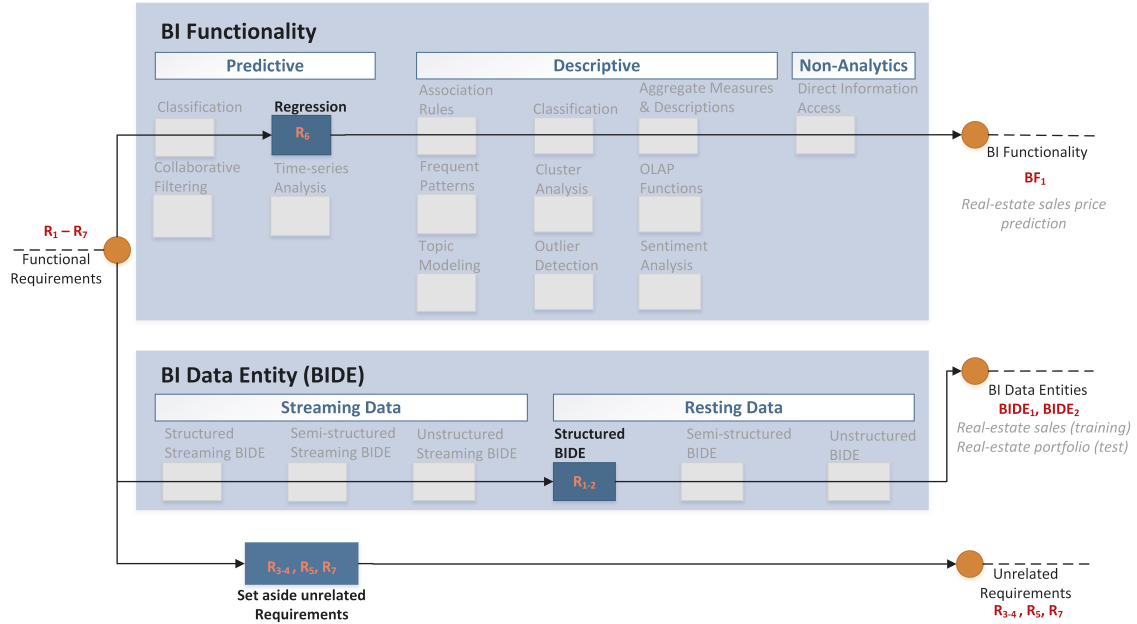

Figure 5.8: Execution of GOBIA.DEV Phase I in the KRE case: Focus on selection of BI functionality and BI data entities.

Table 5.10: BI functionalities and BIDEs for the KRE case derived in GOBIA.DEV Phase I.

Element Type	Identifier	Name	Element
BI functionality	BF_1	Real-estate sales **price prediction**	Regression
BI data entity	$BIDE_1$	Real-estate sales (training)	*Structured* BIDE
	$BIDE_2$	Real-estate portfolio (test)	*Structured* BIDE

Check alignment between BI functionality and BIDE. Set aside requirements R_3 and R_4 refer to data preparation respectively pre-processing functionality, which might be needed between BF_1, $BIDE_1$, and $BIDE_2$. A closer inspection reveals that R_3 refers to technical pre-processing of the data, which is not considered a high-level preparation activity with respect to the GOBIA.REF functional reference architecture (cf. Section 4.2.1). Specifically, this requirement needs to be fulfilled when ensuring that incoming $BIDE_1$ and $BIDE_2$ are made available in a structured format for analysis, regardless of input data, which is always CSV in this case. Therefore, only R_3 necessitates a data preparation step. This requirement refers to feature extraction, which is a subset of methods for dimension reduction (cf. *attribute subset selection* in [HKP11, p. 100]). Therefore, a data preparation step DP_1 *Select relevant features* is added (cf. Table 5.11). Such preparation is only needed for the concluded real-estate sales, which are used as training data in $BIDE_1$.

Check elements alignment to requirements. As R_3 has been set aside for upcoming phases, non-functional requirements need to be checked for alignment. R_8 imposes quality criteria for the analysis result. The selection of regression does not inhibit the fulfillment of this criteria.

Table 5.11: Data preparation step added for the KRE case in the course of GOBIA.DEV Phase I.

Element type	Identifier	Name	Inputs	Outputs
Data preparation	DP_1	Select relevant features	$BIDE_1$	BF_1

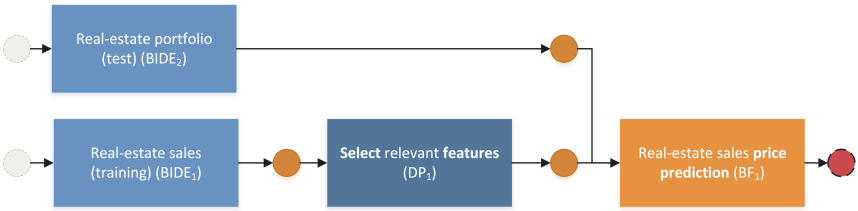

Figure 5.9: GOBIA.DEV for the KRE case: Abstract BI architecture of Phase I depicted using a Petri net with the three element types BIDE, data preparation, and BI functionality ordered from left to right.

Compose Architecture. Figure 5.9 shows the final output of Phase I, the abstract BI architecture. It contains the connections between the three element types created in this phase.

Phase II

Separate by BI functionality and subordinate data preparation and BI data entity elements. The separation into to three basic building blocks is directly attainable. The result from the previous phase in Figure 5.9 allows to derive the three elements immediately.

Add Analytic/Processing/Storage/Generation ABBs. In this case there is only one target BI functionality, one tactical plan TP_1 is created that contains all elements that cumulate into the analytics functionality in BF_1 (cf. Table 5.12).

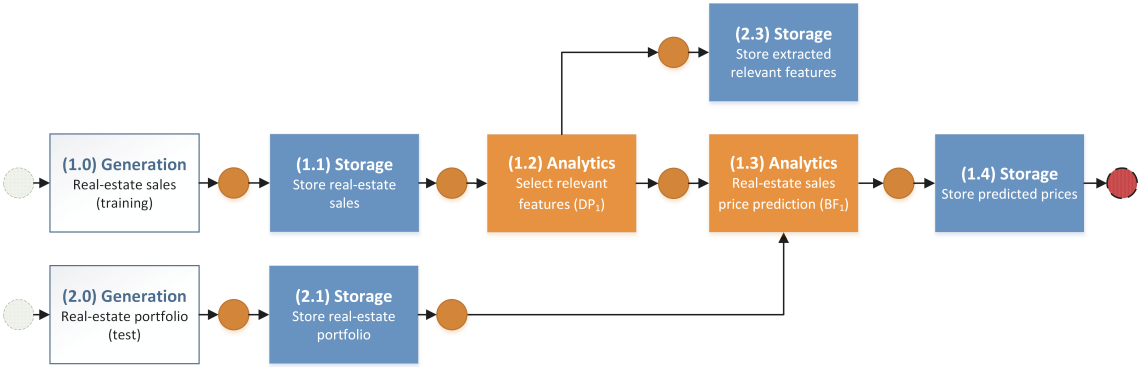

Figure 5.10: GOBIA.DEV Phase II for the KRE case: Tactical plan perspective of the semi-concrete BI architecture, which is the main output of Phase II. A unified view on the tactical plan outlines value generation using ABBs.

Table 5.12: GOBIA.DEV Phase II: Tactical plan for the KRE case.

Identification	Target BI functionality
TP_1	Real-estate sales price prediction (BF_1)

For $BIDE_1$ and $BIDE_2$ data generation ABBs are created, as both are external data to be ingested into the system. Moreover, each of them is mapped to a subsequent storage building block. The reasoning is that incoming data is persisted as structured data needs to be permanently available for several analysis iterations. The data preparation step DP_1 needs to be mapped to an analytics ABBs, because dimension reduction techniques are conducted using analytics functionality. DP_1 is further mapped to a subsequent storage ABB, where the extracted features are stored into to satisfy R_5. Lastly, the main BI functionality is naturally represented by an analytics ABB. Moreover, due to R_7, a storage building block is added afterwards to store the final predicted prices.

Compile into Architecture grouped by Analytics ABBs. Figure 5.10 shows the compiled result focused on the tactical plan TP_1. All mapped ABBs are listed in Table 5.13. Lastly, alignment to requirements and goals is checked. R_5 and R_7 were considered by adding storage blocks. Technical pre-processing as necessitated by R_3 refers to steps typically covered before making the final data available to storage (i. e., data acquisition). Thus, it is handled in Phase III. Considering the goals and usage scenario at hand, no conflicts due to the particular choice of ABBs and tactical plans can be detected, which would necessitate a re-evaluation of the goals. This concludes the second phase. The full semi-concrete BI is depicted by Figure B.7 in Appendix B.

Phase III

Decompose into Architecture Building Blocks and Assign Layers. After a decomposition into ABBs, the layers for the KRE case can be assigned. The two analytics building blocks 1.2 and 1.3 are mapped to the data

Table 5.13: GOBIA.DEV Phase II: List of Architecture Building Blocks for the KRE case.

ABB	Type	Description	Tactical plans
1.0	Generation	Real-estate sales (training) ($BIDE_1$)	TP_1
2.0	Generation	Real-estate portfolio (test) ($BIDE_2$)	TP_1
1.1	Storage	Store real-estate sales	TP_1
2.1	Storage	Store real-estate portfolio	TP_1
1.2	Analytics	Select relevant features (DP_1)	TP_1
1.3	Analytics	Real-estate sales price prediction (BF_1)	TP_1
2.3	Storage	Store extracted relevant features	TP_1
1.4	Storage	Store predicted prices	TP_1

analytics layer. The four storage blocks are mapped to corresponding data storage layers. Lastly, both data generation blocks are mapped to both the generation layer and the acquisition layer. The acquisition layer is necessary to conduct technical pre-processing as defined in R_3. Moreover, the actual input sources are known to be CSV files, which should become structured resting data as specified by $BIDE_1$ and $BIDE_2$.

Select Technology Classes. The selection process from data sources to storage and analytics blocks is as follows:

Data Acquisition Technology Incoming test and training data (1.0 and 2.0) are mapped to semi-structured resting data as the data source are CSV files. Due to their property as resting data, *Data Integration and ETL Tools* are selected for both of these sources. The GOBIA tool repository offers *Pentaho Data Integration*, *Talend Open Studio*, and *Apache Sqoop* for this technology class.

Data Storage Technology To select a storage technology, data volume is measured. The input data size is below 1 MB. As no external

data is considered, the generated and to-be-stored data in each step of the exploratory analysis is suitable for *Centralized Storage*. This applies to all four data storage blocks 1.1, 2.1, 2.2, and 1.4. The data variety of all data is structured. After being acquired, the CSV data should be transformed into a structured table format as specified in Phase I and in accordance to the contents of the dataset. Intermediate and final results of the analysis can also be interpreted as structured data. Predicted prices can be seen as additional attribute to the already structured input data. Extracted features are also in a tabular format, leading to structured storage. The uniform choice for all of this is a *Centralized SQL RDBMS*. Thus, *MySQL* and *PostgreSQL* are emitted from the tool repository.

Data Analytics Technology Both *dimension reduction* (ABB 1.2) and *regression* (ABB 1.3) are considered an *advanced analytics* functionality, which does not fall into the realm of traditional BI analytics. The workload estimation for advanced analytics tool selection is based on the input data size, which is below 1 MB. Even complex regression and dimension reduction techniques do not warrant distributed analytics given this data, which is why *Centralized Analytics* are chosen, which activates *Centralized Analytics Tools*. The GOBIA tool repository outputs *Anaconda (R & Python)*, *RapidMiner (Standalone)*, *KNIME*, and *Weka (Standalone)* for this selection. Next, the capabilities of these are cross-checked with the analytics functionality they should implement. The capability matrix in Table C.4 reveals that all aforementioned tools support both dimension reduction and regression. All choices are summarized in Table 5.14.

Create Compatible Technology Sets. Before the architecture alternatives can be determined, incompatible tool combinations are excluded from the result. The CSV files with the training and test data can be assumed to be on a local hard drive or a locally accessible drive. All three acquisition tools support ingesting data from the local file system and outputting into any RDBMSs (cf. Table C.5). It should be noted that Apache Sqoop is marked

Table 5.14: Technology classes selected for KRE in GOBIA.DEV Phase III.

Tactical plan	Element	Layer	Technology Class Selection
TP_1	(1.0) Generation	Data Acquisition	Data Integration and ETL Tools
TP_1	(2.0) Generation	Data Acquisition	Data Integration and ETL Tools
TP_1	(1.1) Storage	Data Storage	Centralized RDBMSs
TP_1	(2.1) Storage	Data Storage	Centralized RDBMSs
TP_1	(1.2) Analytics	Data Analytics	Centralized Analytics Tools
TP_1	(1.3) Analytics	Data Analytics	Centralized Analytics Tools
TP_1	(2.3) Storage	Data Storage	Centralized RDBMSs
TP_1	(1.4) Storage	Data Storage	Centralized RDBMSs

as only being partially compatible to RDBMSs outputs, as the target table must exist and essentially no pre-processing can take place (cf. [100]). R_3 dictates that data must be pre-processed into its proper from. Partial RDBMS compatibility and no processing capabilities — as documented in the supplementary tool attributes in Table C.1 — leads to the decision to already exclude Apache Sqoop at this point. If less information was available for this decision, it would take place in the detailed technological evaluation after GOBIA.DEV Phase III has concluded. Finally, compatibility between the chosen analytics tools and the RDBMSs is checked. While all four analytics tools can load data from an RDBMS, only *Anaconda (R & Python)*, *RapidMiner (Standalone)*, and *KNIME* are able to write data back to a database. *Weka (Standalone)* can only export results as files and is removed from the result set. Notably, the constraint C_1 attached to R_4 and R_5 should be considered during the post-GOBIA evaluation. However, consulting the documented sources of information for these tools at this point (cf. Table C.4), it can already be concluded that all of them satisfy the constraint and offer various analytics methods for dimension reduction and regression.

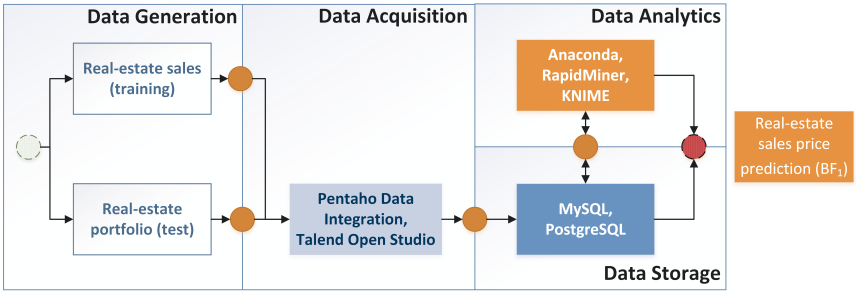

Figure 5.11: GOBIA.DEV Phase III for the KRE case: Simplified illustration of BI architecture alternatives on all layers of Phase III. Data flows are not depicted separately.

The final technology sets are: **(Pentaho Data Integration, Talend Open Studio), (MySQL, PostgreSQL), (Anaconda, RapidMiner, KNIME)**.

Compose BI Architecture Alternatives. Figure 5.11 depicts a combined and simplified view of all aforementioned alternatives into a set of BI architecture alternatives.

5.2.3 Big Data Analytics: Plagiarism Checking

The final case application deals with large scale academic plagiarism checking. Services like iThenticate by Turnitin LLC [385] offer to upload an academic work in question and identify parts of the work that are identical to available documents in the Internet and published scientific works. These duplicate texts in documents help scholars to recognize possible academic misconducts in their own or in other works (cf. [386]). Studies indicate that academic misconduct has been increasing for the last decade (e. g., cf. [FSC12]), although differences between various countries can be identified [Amo14]. Particularly, academic misconduct among students is under discussion since the widespread availability of personal computers and the Internet. Already in 1999, Austin and Brown [AB99] pointed out that widespread

media availability leads to new challenges in detecting misconduct in submitted works of their students. Thus, the topic of finding duplicate text in an academic document is a purposeful and practice-relevant topic for a case application using the GOBIA method.

Aforementioned service iThenticate promises to deliver results on uploaded documents in "minutes time" [385]. This resembles a near real-time response as defined in Section 2.2.6. The corpus of iThenticate consists of more than 50 billion pages from the Internet, where more than 10 million pages are added daily by a web-crawler according to Turnitin LLC [383].

Potthast et al. describe the general approach of so-called *external plagiarism checking* in [PSE+09]. The basic steps start with building a document collection. Using approximate representations, which reduce data size and try to avoid information loss, the documents of the corpus are compared to an input document ("heuristic retrieval" [PSE+09, p. 2]). After a smaller subset of candidate documents has been filtered from the collection, a more detailed comparison to the input document follows.

The goal of this case is to demonstrate how the GOBIA method supports the detection and handling of insuperable challenges and goal conflicts. Here, the case properties necessitate a customized solution using specific algorithms, which is able to fulfill the case conditions. Two extraordinary challenging aspects can be identified. First, the similarity checking for duplicate text passages should be conducted in near real-time in a document corpus, which is similar in size to iThenticate's database with billions of indexed web pages and millions of publications in storage. Next, the management of such a large collection of data easily surpasses that of typical Big Data and other BI systems. In addition to that, the specific algorithms used by iThenticate are not published. Still, finding similar items and extracting text features from documents (cf. [LRU14, pp. 68ff.]) is a complex task — even before considering the data size in this case.

Generally, the detection of insuperable challenges is maintained in GOBIA.DEV. Necessary for this are that conditions for these "break points" can be derived from the goals and requirements. Each phase has a break point, which mandates to cross check the intermediary results with goal and

requirements (Phases I and II) or the consideration of BI architecture alternatives (Phase III and post-GOBIA). This final case application is described using the fictitious company *PlagCheck*, who desire to build a plagiarism detection service similar to iThenticate. In particular, it should offer a comparable scale and alike features. To better compete with iThenticate, a near-real time response time for document similarity checks is desired.

Goals and Requirements. The overall goal of the **PlagCheck case** is to build a BI system, which indexes web-documents, published academic papers, and previously submitted documents. These are used to answer requests to determine duplicate text paragraphs between this corpus and a newly uploaded documented in near real-time. Thus, the **goals** can be formulated as following:

G_1: The system shall acquire access to all web-documents.

G_2: The system shall acquire access to published and uploaded academic papers.

G_3: The system shall provide text-similarities between a user upload in the text corpus in minutes time.

The application *scenario* for PlagCheck is similar to iThenticate. The primary target group are researchers, who would like to verify whether an uploaded document has duplicate text passages with other web or published documents. A web-crawler or similar software works in a distributed fashion and collects documents from both the Internet and from scientific publications. These should be ingested by the BI system and made searchable against a query, which consist of an uploaded document. It is assumed that a corpus of similar size is built (i. e., several billions of web documents, and more than 100 million of publications [384]).

Based on this scenario, the **functional requirements** are as follows:

R_1: The system shall provide access to new web documents.

Description: The system needs to be constantly updated with newly found web documents. In iThenticate's case, there are 10 million new

web documents crawled each day. This is to be expected for PlagCheck as well.
Requires: —
Relates to: G_1

R_2: The system shall provide access to new scientific publications.

Description: The system needs to access new scientific publications. Here, it is assumed that they are made available from journals and publishing partners via an API, which allows to download documents as, e. g., PDF. The number of newly ingested items per day should be significantly smaller than for web documents. However, the overall corpus of documents might still become large. The API is assumed to work with a publish-subscribe paradigm (cf. Section 2.2.6), which allows to immediately ingest new publications.
Requires: —
Relates to: G_2

R_3: The system shall extract text features from ingested web documents.

Description: Crawled web documents need to be stripped from any HTML tags and other non-textual markup in the documents, before computing a characteristic *feature* set from the document text, which describes the document in a compressed way (e. g., shingling [LRU14, pp. 72f.] or based on word-frequencies [274]).
Requires: R_1
Relates to: G_1

R_4: The system shall store web documents text features.

Description: After the textual contents of web documents and their text features have been extracted, the features need to be persisted. Instead of keeping the complete text, only the URI and the compressed extracted features are kept.
Requires: R_3
Relates to: G_1

R_5: The system shall extract text features from publication documents.

Description: The system needs to extract the texts and their features from PDF-based publications to make them properly searchable.
Requires: R_3
Relates to: G_1

R_6: The system shall store publication features.

Description: The system needs to store the textual features of publications in order for the text to become accessible to the BI system.
Requires: R_2
Relates to: G_2

R_7: The system shall provide access to an uploaded document.

Description: An uploaded document by a user should be accessed by the system to calculate the similarity (R_8). User-uploaded documents in PDF can be accessed and ingested from a web-server.
Requires: —
Relates to: G_3

R_8: The system shall extract text features from publication documents.

Description: The systems need to extract the text and its features from PDF-based user uploads to make them properly comparable.
Requires: R_7
Relates to: G_3

R_9: The system shall provide a text similarity report for an uploaded document.

Description: The system shall provide a text similarity report. The report consists of a total similarity percentage, and a list of similar texts with associated duplicate document excerpts.
Requires: R_5, R_6, R_7
Relates to: G_3

Table 5.15: Initial input data items from the PlagCheck case to start the GOBIA.DEV process.

Input	Artifacts from the PlagCheck case
Initial Goals	G_{1-3}
BI Requirements	R_1 – R_{12}

R_{10}: The system shall be a user-accessible web application.

Description: A web-interface should be the system's graphical user interface. All interactions with the BI system take place there.
Requires: —
Relates to: G_3

Additionally, two **non-functional requirements** are necessary:

R_{11}: The system shall provide the similarity rating in near real-time, i. e., within minutes-time.

Description: To be able to compete with iThenticate, the PlagCheck system should be able to answer a user's request for a similarity check in near real-time.

R_{12}: The system shall make new documents accessible for similarity ranking in near real-time.

Description: While documents are assumed to enter the system irregularly, i. e., as soon as crawlers discover them, they need to be made available for similarity ranking quickly in near-real time to ensure that recent documents are considered.

Phase I

GOBIA.DEV is started with ten initial functional requirements, two non-functional requirements, and three goals as listed in Table 5.15.

Table 5.16: PlagCheck case: Separation of input data into GOBIA.DEV Phase I into functional and non-functional requirements as first step.

Requirement Type	Requirements from the PlagCheck case
Functional Requirements	R_1 – R_{10}
Non-Functional Requirements	R_{11} – R_{12}

Extract Functional and Non-Functional Requirements. First, the functional and non-functional requirements are disambiguated according to the distinction above. The result is noted in Table 5.16.

BI functionality and BI data entity definition. Next, the functional requirements are used to determine BI functionalities and BIDEs.

BI functionalities. The core functionality of the service is defined by R_9: the detection of similarities between documents (cf. Table 4.5). Measuring the similarity between one document and several is a *clustering* functionality at core, albeit heavily dependent on the similarity *measure* (cf. [274]). If a text excerpt in a document is in the same cluster as another document according to the chosen similarity measure, the texts are likely to be similar or even duplicates. Thus, BF_1 is created.

BI data entities. Three different data entities can be identified. First, web documents entering the system ($R_1 \rightarrow BIDE_1$), then the scientific publications ($R_2 \rightarrow BIDE_2$), and the uploaded document ($R_7 \rightarrow BIDE_3$). All texts are unstructured and resting BIDE. Notably, the web documents are passed on by the web-crawler for ingestion as soon they are scraped, which renders them streaming data. Although the number of crawlers is under PlagCheck's control, a large number is needed to index millions of documents each day. As they run constantly and output scraped web documents, a selection of a streaming BIDE is appropriate. The same applies to scientific publications, although the velocity is lower, because there are fewer documents in total.

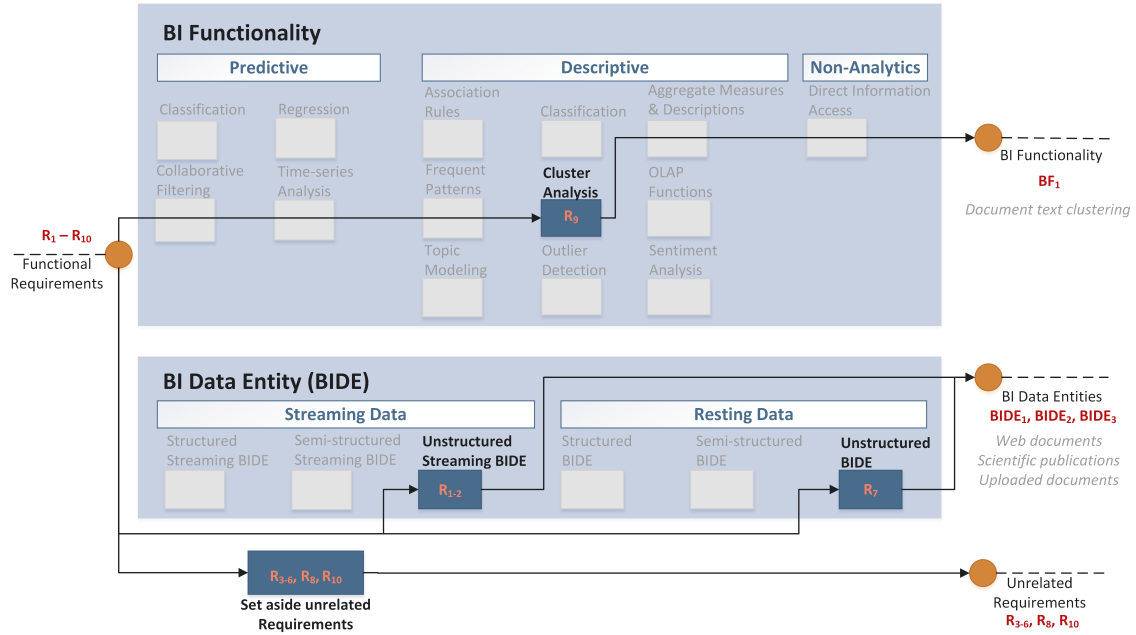

Figure 5.12: Execution of GOBIA.DEV Phase I in the PlagCheck case: Focus on selection of BI functionality and BI data entities.

Table 5.17: BI functionalities and BIDEs for the PlagCheck case derived in GOBIA.DEV Phase I.

Element Type	Identifier	Name	Element
BI functionality	BF_1	Document text **clustering**	Cluster Analysis
BI data entity	$BIDE_1$	Web documents	*Unstructured Streaming* BIDE
	$BIDE_2$	Scientific publications	*Unstructured Streaming* BIDE
	$BIDE_3$	Uploaded documents	*Unstructured* BIDE

Set aside requirements. The functionalities to extract text and features from documents constitute typical preparation functionalities and are set aside at first (R_3, R_5, R_8). Moreover, persistence-related requirements are not needed yet (R_4, R_6). Also, the requirement for the BI system to be a web application with a web-interface (R_{10}) is also set aside as it is not related to the definition of BI functionalities and BIDEs.

Check alignment between BI functionality and BIDE. The previously set-aside requirements concerning data preparation lead to several data preparation steps here. The extraction of text first and the subsequent extraction of text features can be regarded as a *Dimension Reduction* task. The extraction of features from a text is a form of dimension reduction, which attempts to minimize information loss (cf. Section 2.3.3). Before that can be done, text from web document and regular PDF documents (both uploaded document and scientific publications) need to be extracted using a *Filter* activity.

Check elements alignment to requirements. The first alignment of the created elements to the requirements could yield several *hazards* that are inherent to this solution, which is the size of the data involved and that

Table 5.18: Data preparation step added for the PlagCheck case in the course of GOBIA.DEV Phase I.

Element type	Identifier	Name	Inputs	Outputs
Data preparation	DP_1	Extract web document text	$BIDE_1$	DP_3
	DP_2	Extract document text	$BIDE_2$, $BIDE_3$	DP_3
	DP_3	Extract text features	$BIDE_{1-3}$	BF_1

essentially all data items participate in the cluster analysis — albeit using only a subset of data. In addition, the fulfillment of the NFR R_{11} regarding the analysis latency of the solution might be particularly at risk due to this. Nevertheless, the demonstration continues at this point, as the issues become more apparent when storage is involved in the next phase.

Compose Architecture. The compiled result is depicted in Figure 5.13.

Phase II

Separate by BI functionality and subordinate data preparation and BI data entity elements. In this case, one main BI functionality BF_1 yields one tactical plan TP_1 *Document text clustering (BF_1)*, which depicts the necessary data and preparation steps for BF_1.

Add Analytic/Processing/Storage/Generation ABBs. As cluster analysis in BI functionality BF_1 is an analytics functionality, an analytic building block 1.4 is added. The text extraction preparation steps DP_1 and DP_2 yield a respective processing ABB (1.1 and 2.1). The three data entities $BIDE_{1-3}$ can be directly mapped to respective generation blocks 1.0, 2.0, and 3.0 as no storage is required for their raw form and the conceptual BIDEs accurately represent their actual data sources. Notably, the feature extractor DP_3 that utilizes dimension reduction needs to be mapped to the analytics building block 1.2, as the extraction of text features relies on appropriate analytic

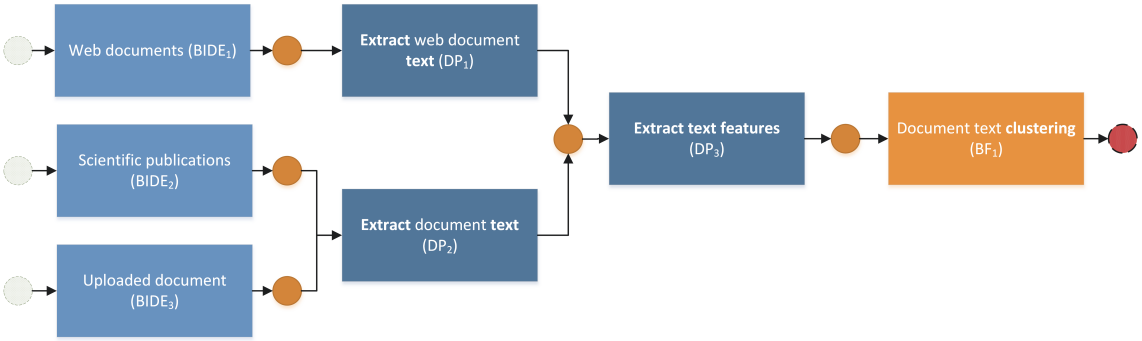

Figure 5.13: GOBIA.DEV for the PlagCheck case: Abstract BI architecture of Phase I depicted using a Petri net with the three element types BIDE, data preparation, and BI functionality ordered from left to right.

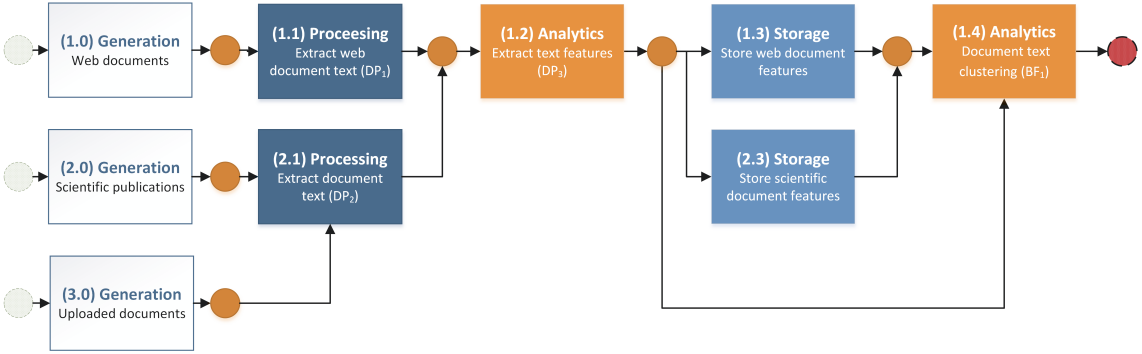

Figure 5.14: GOBIA.DEV Phase II for the PlagCheck case: Tactical plan perspective of the semi-concrete BI architecture, which is the main output of Phase II. A unified view on the tactical plan outlines value generation using ABBs.

Table 5.19: GOBIA.DEV Phase II: List of Architecture Building Blocks through all PlagCheck tactical plans.

ABB	Type	Description	Tactical plans
1.0	Generation	Web documents	TP_1
2.0	Generation	Scientific publications	TP_1
3.0	Generation	Uploaded documents	TP_1
1.1	Processing	Extract Web document text (DP_1)	TP_1
2.1	Processing	Extract document text (DP_2)	TP_1
1.2	Analytics	Extract text features (DP_3)	TP_1
1.3	Storage	Store Web document features	TP_1
2.3	Storage	Store scientific document features	TP_1
1.4	Analytics	Document text clustering (BF_1)	TP_1

methods. Additionally, the extracted features for Web documents and scientific publications should be persisted to create a large feature corpus (R_4, R_6) so that two respective storage ABBs (1.3 and 2.3) are added. Conjointly with not persisted, extracted features from uploaded documents, the stored features can become subject to the cluster analysis to identify similarities in ABB 1.4. The resulting building blocks are summarized in Table 5.19.

Compile into Architecture grouped by Analytics ABBs. A compilation including storage ABBs (cf. Figure 5.14) results in a clearer picture of the previously identified hazards.

The refinement of the abstract BI architecture through GOBIA.DEV adds details to crucial aspects of the architecture, which aid in identifying more specific mis-alignments to the input goals and requirements:

1. **Details regarding BI functionality and data preparation.** By refining the abstract BI architecture and mapping generic *preparation functionality* onto more specific, technical building blocks, the distinction between generalized processing and more complex analytics functionality is aided (cf. Section 4.3.2).

2. **Details regarding data storage.** The explicit consideration of persistence as mandated by the requirements helps to ingrain the desired architectural usage pattern into the semi-concrete BI architecture. However, this also reveals if storage might induce delays, which contradict latency requirements such as data or analysis latency (cf. Section 2.2.6).

The addition of two storage building blocks exemplifies the goal conflict that is created by requiring low analysis latencies and requiring large-scale distributed persistence of data. Additionally, a potentially complex text similarity matching using a cluster analysis with text features is needed. Moreover, through refinements provided by GOBIA.DEV, the text extraction feature is identified as (complex) analytics functionality, highlighting that two subsequent analytics functions have to be applied on that large corpus of input data. Simultaneous persistence of unusually large data given near real-time requirements (R_{11}) outlines a goal conflict, which leads to a misalignment of requirements and the architecture. The solution of such misalignments before continuation of the next phase is mandated by GOBIA.DEV Phase II. Here, the end place *To-be aligned goals* (cf. Figure 4.8) is reached and the re-evaluation of goals is conducted next.

Re-Evaluation of Goals and Discontinuation

The step for re-evaluation and possible refining of goals (cf. Figure 4.1 in Section 4.1.2) is started if a conflict between created elements in an architecture in GOBIA.DEV Phase I, II, or III is detected. While GOBIA.DEV Phase I includes a refinement of a requirements during the phase to allow the creation of appropriate BI functionalities, GOBIA.DEV Phase II directly triggers a goal re-evaluation.

The guiding question in this step is whether an update to goals and — by extension — requirements is sufficient to overcome the detected misalignments.

Here, a re-evaluation of goals yields that requirements can only be harmonized with the artifact if more precise knowledge with respect to the

algorithm employed is available. In such special cases, efficiently designed algorithms directly aligned to potentially used technology have to be selected and implemented (e. g., cf. [LRU14]). Although the GOBIA method offers standardized building blocks for a broad range of use cases, bleeding edge systems that rely on overcoming specific challenges with algorithms that offer a competitive advantage might rely on knowledge, which is not part of such common models and methods. Thus, the process is *discontinued* for PlagCheck at this case.

5.3 Evaluation Results Discussion

Finally, the results of the three case applications in this chapter and the previously conducted FROG AIR case are examined. First, the results of each case are briefly summarized in Section 5.3.1.

Based on these results and the construction rationale of the method (cf. Chapter 4), the fulfillment of solution artifact requirements is reflected upon (cf. Section 3.7.2) in Section 5.3.2.

5.3.1 Summary of Results

Four cases were used to illustrate the GOBIA method. They were set up with a brief case motivation and background as well as initial goals and requirements to start GOBIA.DEV. Their results are briefly summarized in the following.

FROG AIR case results. The FROG AIR case (cf. Section 4.3[2]) features airline company FROG AIR, a customer of MIDAS. MIDAS builds a Customer Service Monitor to examine interactions on Facebook and Twitter using aggregate metrics, complemented by a sentiment analysis. This represents a mixture of a traditional dashboard with metrics and novel social media data and analyses. In GOBIA.DEV Phase I, it was demonstrated how requirements (R_{16} and R_{17}) can be added if the formulation of necessary elements is not

[2] Case description located in Section 1.3; goals and requirements in Section 2.5.4.

yet possible. Here, two requirements mandating the preparation of data before use, were created. With respect to the GOBIA.REF functional reference architecture, GOBIA.DEV Phase I lead to the selection of two types of BI functionalities (*Aggregate Measures & Descriptions* and *Sentiment Analysis*) as well as one type of BIDE (*Semi-structured BIDE*). During GOBIA.DEV Phase II, three additional points for storage were added: first, after ingesting data from the social networks; second, after pre-processing and combining the data and, third, to store the results of the sentiment analysis. The three BI functionalities were technically disambiguated into two processing and one analytics building block. In GOBIA.DEV Phase III the suitable mix of technologies included traditional data integration tools, NoSQL data stores, and centralized analytics tools. In consideration of the overall solution space in the GOBIA.REF technological reference architecture, only a small subset of technologies remains as result. More specifically, out of 25 building blocks in total, five remain for the FROG AIR case (cf. Section 5.15). The chosen tool combination exhibits a mixture of traditional technology such as data integration tools, but also novel ones such as NoSQL data stores.

HPL case results. The HPL case (cf. Section 5.2.1) expanded upon the idea of tracking movable assets in a hospital and determining usage patterns and violations of allowed usage boundaries (geofencing). While the desired aggregate measures are traditional uses of data, its nature called for the use of Streaming Processing Technologies. GOBIA.DEV Phase I for the HPL case showed that functionality needed for this case can be summarized under *Aggregate Measures & Descriptions*. Moreover, two requirements lead to *Direct Information Access*, where prepared data is presented to users for decision making. Besides semi-structured streaming BIDEs, master data regarding assets was considered structured data. The second phase demonstrated that a conceptual BIDE element is not necessarily the data that is generated and "enters" a BI system. Here, asset location data is derived from pre-processing data in the location event stream. This leads to a separate storage building block for that data. The semi-concrete BI architecture also clarifies that several sources of data are used conjointly to realize functionality expressed through the five tactical plans. The technology selection through GOBIA.-

DEV Phase III resulted in two sets of tools. First, stream processing tools, i. e. SPEs and data collection tools, were selected for the location event stream pipeline. These cover all activities from acquisition to final processing as typical for stream processing systems (cf. Section 3.6). To persist an asset location history, data is diverted into structured storage. Master data from external sources is acquired by ordinary ETL tools. Computing the measures in most of the cases was simple enough to consider both SQL and NoSQL data processing, whereas the former remained as final choice. With respect to the combination of technologies, the HPL case also had a combination of both traditional and novel choices, although the streaming pipeline is mostly separate. As this case is more diverse than FROG AIR, 8 of 25 building blocks from the GOBIA.REF technological reference architecture were selected (cf. Section 5.15).

KRE case results. The KRE case (cf. Section 5.2.2) covers typical exploratory analyses. Based on a Kaggle challenge, housing data on conducted sales should lead to a price prediction model to estimate prices of yet unsold real-estate. Environments to conduct exploratory analyses are a typical approach to induce new knowledge, in particular if the business question is not or only partially known. This led to a particular constraint on the tool emission process, which should only consider tools with several algorithms for each required analytics functionality. The abstract BI architecture with data entities, preparation steps, and BI functionality as result of GOBIA.-DEV Phase I consists only of a few selected elements, which underlines that value is created while finding the proper prediction model. GOBIA.DEV Phase II detailed the data pipeline, which leads the housing data into the exploratory system, resulting in structured data for analysis. Therefore, the selected storage and acquisition technology in this case is traditional. The selection of an advanced analytics tool demonstrated the capability-based tool emission, where BI functionality needs to be covered by the requested analytics tool. Out of 25 building blocks in the GOBIA.REF technological reference architecture, four building blocks are selected for the KRE case (cf. Figure 5.15).

PlagCheck case results. The last case, PlagCheck (cf. Section 5.2.3), is based on the plagiarism detection service iThenticate. The goal was to attempt to imitate that service using standardized building blocks and generic knowledge about the text similarity detection process. It is purposefully constructed to demonstrate how the checks for alignment using the created architectures and elements can support the detection of goals and requirements conflicts (misalignments). GOBIA.DEV Phase I already hinted at potential alignment hazards due to the complexity of the involved functionality, the data sizes involved, and the near real-time requirements. In GOBIA.DEV Phase II, the situation became more apparent due to the refinements conducted there. The more detailed view on functionality blocks and explication of data storage supported the business decision to discontinue the process at this stage.

5.3.2 Reflection on the Solution Artifact Requirements

An evaluation in the DSRM process provides a more formalized means to compare the desired objectives with the actually reached ones (cf. Section 1.2). What separates the evaluation of an artifact from its demonstration is that the latter only requires "effective knowledge" about the artifact and its usage to solve a specific problem. An evaluation adds to this by specifically observing the outcomes and determining "how well" it supports the solution of the outlined issues [PTRC07, p. 56]. Peffers et al. note that a qualitative evaluation using logical argumentation can be an appropriate means for an evaluation, depending on the research characteristics. Particularly, the evaluation could be conducted "as a comparison of the artifact's functionality with the solution object" [PTRC07, p. 56]. This approach is applied here, because combining illustrative scenarios with logical arguments while constructing the method allows to qualitatively argue if the results can be *rationally* considered to fulfill the solution artifact requirements (cf. Section 3.7.2).

$\mathbf{SR_1}$ states that the solution should consist of a BI reference architecture and a development process as stated in the research question. The fulfillment of this requirement is not dependent on case applications. By elaborating

upon the rationale while constructing these two parts of the GOBIA method, it is clarified that both a BI reference architecture, represented through a high- and a low-level reference architecture, and a development process are part of the method.

SR_2 defines that a "current" selection of both novel and traditional technologies and approaches is part of the BI reference architecture. The choice of basic building blocks described in Chapter 2 is supported by the analysis of analytical architectures in Chapter 3 and in particular the NBDRA [NIS15b], which has the goal to standardize the solution spectrum of Big Data solutions and can be amended by traditional technologies. As the examined publications there are no more than ten years old, it can be assumed that the condition of "current" technologies is met. Section 4.2.2 elucidates and justifies the particular structure and selection of technologies, which form the basis of technology selection. Figure 4.3 visually indicates the combination of traditional and novel elements. Moreover, as also stated in SR_2, the construction of the BI reference architecture deliberately referred to the unified Big Data value chain in Section 3.2.2 to define its structure and aid the choice of technologies and approaches.

As the selected technologies also depend on the BI functionalities and data in the GOBIA.REF functional reference architecture, which realize the requirements, a representative selection for this had to be formatted as well in order to allow a precise mapping of case requirements. This was conducted by considering various background literature on Big Data and Big Data analytics (cf. Section 2.2.1 and Section 2.3). Notably, the Big Data reference architectures examined in Chapter 3 do not depict specific functionalities in the level of detail and in the number as the GOBIA.REF functional reference architecture does.

SR_3 specifies that the technology selection should ultimately be based on goals and requirements from a given use case. Generally, GOBIA.DEV Phase I enforces an explicit connection to given goals and requirements and cannot start without those. This mapping worked for the four use cases, although the underlying base functionalities such as classification or regression have to

be derived from requirement descriptions. A sample mapping support table (cf. Table 4.5) was provided to alleviate this still manual process, although it is not intended to be comprehensive. Through the subsequent Phase II and Phase III, which further refine the abstract BI architecture to a finally customized BI architecture, technology selection is conducted. Each phase is connected explicitly by the Petri net process models of GOBIA.DEV. Additionally, the case applications illustrate that how various requirements lead to different selections of BI functionalities, which ultimately meant that a different set of technologies needed to be selected. For instance, in the FROG AIR case, a sentiment analysis was selected as required. In the HPL case, the technology selection was specifically driven by the need to calculate aggregate measures on streams. These measures were explicitly mandated by a specific requirement, while streams were implied by another requirement and the case description

SR$_4$ details how exactly SR_3 should achieve the connection between goals, requirements, and selected technologies. In particular, a BI reference architecture should be explicitly utilized while forming the connection from goals and requirements to selected technologies. The overall flow of the GOBIA method (cf. Figure 4.4) is specifically designed for this requirement, as this is one of the aspects that has been determined as lacking in existing Big Data reference architectures (cf. Section 3.4.7). To achieve this, the GOBIA.-REF functional reference architecture and the GOBIA.REF technological reference architecture are directly embedded into the development process GOBIA.DEV. This integration has the goal to ensure an explicit connection between goals and requirements, the functionalities and data that can realize them, the architectural usage patterns that refine the connections between functionalities and data, and the technology building blocks, which are suitable to realize that patterns given the goals and requirements and the functionality. The four case applications illustrate a range of use cases, for which the GOBIA.DEV process execution was documented and illustrated using figures that depict the taken case-specific decisions.

SR_5 mandates that the selection of technologies should conduct an actual selection by means of reducing the overall solution space in accordance to the requirements. Generally, the choices in GOBIA.DEV Phase III hinder the simultaneous activation of all technology choices on one layer and clear incompatible combinations from the result set. The four case applications illustrate that — despite conducting a separate technology selection for several ABBs — the solution space could be restricted greatly. Four, five, and eight out of 25 building blocks from the GOBIA.REF technological reference architecture are selected respectively (cf. Figure 5.15), i. e., between 16 and 32 % of building blocks remain. Naturally, these four cases do not represent a formal validation that the selectivity cannot become higher at any circumstance. However, the combination of selective choices, the enforcing of compatibility between conjointly used tools, and the results of the case applications indicate that the selection process works as required by SR_5.

SR_6 refers to an independence of particular development methodologies. Although these stem from software and system development practices, they illustrate different approaches to deal with requirements before conducting a project, specifically traditional approaches, where all requirements are specified before implementation starts, and agile approaches where such information is generated "ad-hoc" and iteratively. One of the goals of the three phases of GOBIA.DEV is to request the minimum amount of information necessary to be executable. This reasoning is generally compatible with agile methodologies. Moreover, starting GOBIA.DEV Phase I with more information than necessary using a traditional approach. Here, the case application used a simplified template to document goals and requirement with identity, description, and relationships only. However, a conclusive answer to independence from development approaches can only be given by means of an empirical evaluation in an organizational context.

SR_7 demands that an "appropriate" modeling language and techniques are used, which are able to comply with qualitative model assessments. As briefly outlined in Section 2.4 process modeling in general is an appropriate technique to visualize a method. Specifically, Petri nets, which are capable of

modeling such, exhibit some favorable characteristics such as tokens, which can be interpreted as objects carrying information and begin transformed by GOBIA.DEV. Apart from that, the separate depictions of GOBIA.REF have been conducted for illustrative purposes as the models are also embedded into GOBIA.DEV. Certain deviations from "pure" Petri nets deviations were made (cf. Section 2.4) to increase clarity and decrease complexity of the models. However, a quantitative evaluation using process, respectively modeling quality metrics is still outstanding.

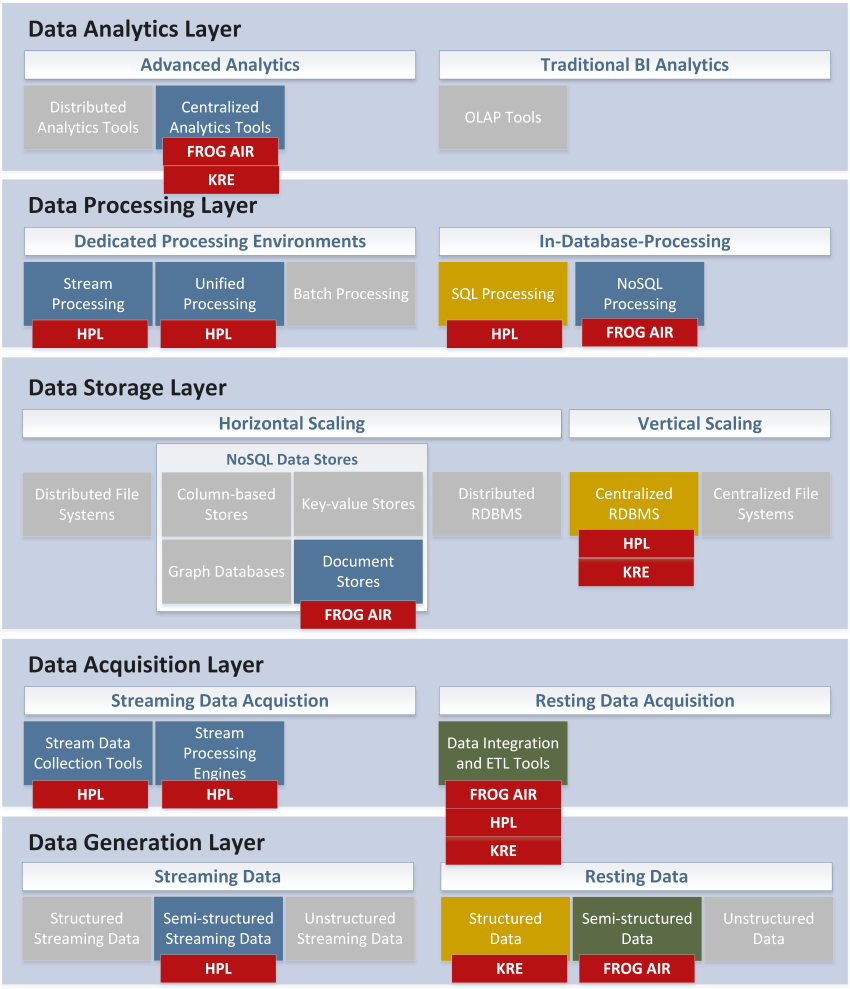

Figure 5.15: Utilization of the elements of the GOBIA.REF technological reference architecture as result of case applications.

6 Discussion

After the evaluation and its results with respective to the solution artifact requirements have initially been examined in Section 5.3, the overall work and its approach is discussed in the following. First, it is stated how the resulting solution artifact, the GOBIA method, addresses the research questions of this work. Next, the overall approach and the evaluation methods employed are reflected upon. Finally, limitations of this work and its approach are outlined.

Relation of the solution artifact to the research questions. The overall aim of this work is to answer research questions RQ_1 and RQ_2. This is done using the GOBIA method as solution artifact. RQ_1 refers to the role of a BI reference architecture in supporting the construction of a customized BI architecture. The GOBIA method addresses RQ_1 by embedding the proposed BI reference architecture — GOBIA.REF — directly into the development process which tackles RQ_2. The analysis of analytical architectures in Chapter 3 shows that the examined Big Data reference architectures do not explicate the connection between the selection of technologies, the (implicit) criteria used for this, and the proposed reference architecture models themselves. Thus, the approach of the GOBIA method is to explicitly integrate the technology selection criteria conjointly with a BI reference architecture into an overall development process, explicating the connection between those parts. The underlying assumption is that this integration allows for the necessary support of the construction of a customized BI architecture by means of a BI reference architecture according to a given use case as demanded by RQ_1.

RQ_2 is addressed by the development process GOBIA.DEV of the GOBIA method. GOBIA.DEV forms a connection between a given case and selected technologies using a three-phase process. First, BI requirements are related

to high-level functionality and data. Based on this, the use case requirements are used to further refine this abstract BI architecture by adding the optional element of storage and mapping functionalities to technical building blocks. In this way, diversified architectural usage patterns can be ingrained. Finally, these building blocks are used to select technologies from respective technology layers, whereas requirement-driven decision criteria inside the process guide selection using technology classes to determine appropriate and compatible tools as instances of technology. These form customized BI architecture alternatives, which are the final result of the GOBIA method. These alternatives are then subject to a more detailed evaluation inside an organization.

Reflection of the approach. As stated in Section 5.1, a knowledge base built on a relevant analysis in the Information Systems domain is the foundation for logical arguments that are employed for an evaluation using illustrative scenarios.

The analysis of relevant technological artifacts is conducted in Chapter 2. The results are mirrored by the analysis in Chapter 3, i. e., the technologies initially introduced can be identified during the analysis as well. Hence, it can be assumed that the identified technological building blocks constitute an appropriate corpus for a knowledge base, which is the foundation for the method rationale of the GOBIA.REF technological reference architecture. The analytics methods which are part of the GOBIA.REF functional reference architecture are established in Section 2.3.3 using several literature sources. However, the analyzed architectures do not depict analysis components in the same detail as eventually used in GOBIA.REF. Thus, the first knowledge base should contribute better to logically arguing in favor of goal attainment of the method than this second.

The four case applications illustrate the applicability of the GOBIA method ([PRTV12, p. 402]). The FROG AIR, the HPL, and the KRE case show how technologies are selected based on goals and requirements as input. In all cases, a selection of technologies is made, which narrows the solution space significantly for these cases. The PlagCheck case demonstrates an exit point in case of maligned requirements. The qualitative assessment in Section 5.3.2

with respect to the fulfillment of the solution artifact requirement can be regarded as being in support of fulfillment. However, this support relies on the illustrative power of the case examples and the argumentative power of established knowledge bases. The general structure of the GOBIA method is as mandated by those requirements: It has a BI reference architecture and a development process.

In summary, the GOBIA method is a suitable answer to the posed research questions in the context of the evaluation in this work. The chosen evaluation method generated results, which are generally in support of this suitability. However, several limitations to this approach can be identified, which hinder the applicability of the method beyond the used sample cases and outside the scope of this work.

Limitations

The limitations that inhibit generalizability of the suitability and utility of the GOBIA method are as follows:

Methods and lack of empirical evaluation. While illustrative scenarios are often used and might allow for indications about generalizability and even be based on real-world problems, they have various limitations. First, they are solely based on logical reasoning and the illustrative power of the employed scenarios. They rely on the fact a proper knowledge base is built and utilize logical argument to assess fulfillment of solution objectives. Here, the case applications are based on real-world examples, but are ultimately synthetic as the specifics are constructed to build the case for evaluation. Thus, it is unclear how the GOBIA method handles independently created case applications, although the current evaluation provides arguments supporting this. Moreover, an empirical evaluation to provide external validation to the claim about utility and applicability is lacking. A case study observes the particular application of a method and studies its effects. While generalizabiliy is lower when one case is examined, empirical results can provide real-world content [PRTV12, p. 408]. As the GOBIA methods aims to solve a real-word problem, such context is generally desirable to argue for further

utility. Other empirical methods include experiments, e. g., in a laboratory setting [PRTV12, p. 402]. Additionally, Prat et al. suggest to use kernel theories from behavior science and to formulate a testable hypothesis about the artifact in question [PCWA15, p. 232].

Comprehensiveness of used literature. The literature search used for Section 3.4 and Section 3.5 is guided by scientific guidelines ([VSN+09]), but does not constitute a structured literature review on the topic. While the identified Big Data reference architectures and architecture implementations are consistent with the theoretical findings, the identified Big Data reference architectures and implementations cannot be regarded as comprehensive. A comprehensive review would strengthen the chain of arguments for the present evaluation.

Selection of case applications. The selected case applications only cover a subset of the technologies that can be selected by GOBIA.DEV (cf. Figure 5.15). Although the cases show that traditional and novel technology can be combined, several parts of the GOBIA.REF technological reference architecture do not directly participate in the evaluation or demonstration. Although the case evaluation supports the rationale of the method, an illustration of these parts of the model is lacking, which inhibits the generalizability of the method.

Execution of model iterations. Although several executions of the GOBIA method are shown, they constitute only one iteration of method, which is designated as iterative. Thus, it cannot be assessed to which extent the GOBIA method works iteratively.

Assumptions for the selection of technologies. The selection process generates results in form of pre-existing software tools, which can implement a desired functionality such as a data storage. Further, to be included in the set of selected technologies for the customized BI architecture alternatives, two tools emitted from the tool repository must be directly compatible to each other.

7 Conclusion

To conclude this work, a brief summary is provided in Section 7.1, which reflects on the background and motivation of the work and summarizes the contributions of each chapter. This is complemented by an overview of possible areas for future research opportunities outlined in Section 7.2.

7.1 Summary

The advent of Big Data in the last decade has spurred the development of a plethora of novel technologies such as NoSQL data stores, Apache Hadoop, MapReduce, and Apache Spark. These technologies are able to utilize data of huge size, large variety, and high velocity. Before that, traditional technologies such as Data Warehouses and Relational Database Management Systems were regarded as almost-universal solution for Business Intelligence, which is a holistic approach to decision-making support in organizations. While a DWH reference architecture enables the development of customized Data Warehouses, no universal BI reference architecture exists that integrates both Big Data and traditional solutions. However, the selection of technologies in this situation is much more complex, due to large number of technologies and the variety of supported use cases. For this reason, two research questions were formulated to determine how a customized BI architecture for a given case can be constructed under these contingencies. These research questions were addressed using a design science research method which resulted in a solution artifact, the GOBIA method as contribution of this work. To support this goal and comply with the chosen research method, the specific structure of the work was chosen.

Chapter 2 was designated to give an overview of the most important building blocks to the both the traditional as well as the novel technology landscape related to BI. This overview started with details of RDBMSs, SQL, and DWHs as traditional BI technologies. It was followed by a selection of representative Big Data technologies: NoSQL data stores, the Hadoop ecosystem, in particular HDFS and MapReduce. The next block of technologies were related to stream processing and data collection, followed by aspects to advance traditional technology. This was completed by elaborating upon traditional and advanced forms of data analytics, highlighting novel approaches such as machine learning, and new domains for analytics. Additionally, several key definitions, characteristics, and aspects of several technologies could be outlined, which supported the later part of the work. Working definitions were outlines for both BI and Big Data, whereas the latter was also characterized using the "5Vs". For both NoSQL data stores and Streaming Processing Technologies, key characteristics were synthesized, which allow to better disambiguate various tools in the various classes. For Big Data analytics, a sighting of selected literature allowed to create an overview of important methods aimed to harmonize new advanced analytics with traditional OLAP analyses in a common format. The chapter concludes with an overview of process modeling, requirements engineering, and a framework for architecture analysis (TORE). Process modeling with Petri nets is briefly introduced as means to later depict the contributed GOBIA method. Requirements engineering provides the definition of goals and requirements and their relationships, which become an important input to the development process GOBIA.DEV. The TORE analysis framework was established to picture four perspectives in an organization to view and analyze architectures.

Chapter 3 aimed to further motivate the research according to the design science process, but also to build up the necessary information to formulate the requirements for the GOBIA method. For that reason, the chapter examined the term architecture and explained the parts of reference architecture, namely reference models and architectural patterns. With this knowledge in mind, the topic of Big Data value chains was elucidated. Value chains

describe the process of value generation by turning data into usable information by means of various activities such as processing and analysis. Based on an overview of different value chains found in academia and practice, a proposal for a unified Big Data value chain was formulated, which should cover both traditional and novel approaches. This unified value chain was subsequently used to characterize various aspects of architectures. After that, an analysis of (reference) architectures was conducted. First, possible advancements to DWH architectures with and without Big Data were highlighted to demonstrate that a combination is feasible and enables a variety architectural usage patterns, which are more diverse than in the traditional DWH reference architecture. Utilizing a literature search on Big Data architectures, six exemplars of Big Data reference architectures were selected, presented, evaluated, and analyzed. The goal of this analysis was to outline and assess key characteristics of Big Data reference architectures with respect to the research goals such as the relation to technological choices and level of detail used for functionality and data. The results included that there exists no universal BI reference architecture among those reference architectures and that technologies were only implicitly considered, strengthening the motivation to contribute to this topic. Following an analysis of technology use and usage patterns in three selected architecture implementations, several technological selection criteria for technology in analytical architectures could be synthesized based on the preceding analyses. These criteria could be related to respective technology classes and capabilities, and aimed at velocity, variety, and volume challenges. Finally, the requirements for the solution artifact, the GOBIA method, could be defined as mandated by the design science process. The analysis confirmed that the required artifact was needed to achieve those goals, which were outlined by the research questions. Moreover, these requirements detailed the basic ingredients that allowed to create a connection between goals and requirements, a BI reference architecture, and the process of technology selection.

The GOBIA method proposal was constructed, explained, and applied to the FROG AIR running example as illustrative scenario in Chapter 4. It presents a crucial part of the design science process, which is the design and

development of the contributed solution artifact. Guided by method engineering as approach to design the artifact, the GOBIA method was placed into an organization context, detailing its use therein, while highlighting the iterative nature of the method. It comprises a development process, which uses goals and requirements and has three phases of refinements and the embedded reference architecture GOBIA.REF to guide the selection of technologies. The reference architecture GOBIA.REF actually consists of a functional and a technological variant to capture both abstract functionalities as well as technologies. The outcome of the method is a set of suitable, customized BI architectures comprised of several tool alternatives for the selected technologies. The rationale for the construction of GOBIA.REF was grounded in the work in Chapter 2 and Chapter 3. The established foundational technologies could be ascertained in their architectural usage and structured according to their roles in the architecture, whose layers were derived from the unified value chain proposal. Part of GOBIA.REF were not only abstract technology classes in the model, but a list of representative software tools for each of them. These were compiled into the GOBIA tool repository, which lists these tools and characterizes them by decision-relevant and supplementary attributes. The functional GOBIA.REF model captures the notion of abstract BI functionalities and data entities, related to analytics and data, which were established in Chapter 2 and realize given goals and requirements. During the elicitation of the three phases of GOBIA.DEV, the FROG AIR case was used to demonstrate the use of the artifact. GOBIA.DEV uses the functional GOBIA.REF to capture functionalities, which are amended with storage and technical building blocks, which allow to relate the intermediate results to concrete technologies of the technological reference architecture in the third phase. Using the GOBIA tool repository, not only capability-based tool selection is conducted, but also an elimination of incompatible tool combinations. The chapter continues with a short overview of post-GOBIA activities, which include the detailed evaluation of the provided alternatives, using more technical factors not part of GOBIA.DEV such as throughput or CAP requirements. These are complemented by organizational, regulatory, and economical aspects. Finally, as the technology landscape is likely to evolve

in the future, extension points to the GOBIA method and tool repository are explained and illustrated.

In addition to designing and developing a solution artifact and demonstrating its use, the DSRM process stipulates an evaluation of the artifact, which was conducted in Chapter 5 and used three additional *illustrative scenarios*, to which the GOBIA method was applied. All three cases were synthetic, but rooted in practical applications and focused on showing different aspects of analytics such as streaming or exploratory analyses. The third case application was particularly used to demonstrate how the GOBIA method supports a user in detecting intermediate results that exhibit a goal and requirement conflict. The evaluation was finalized by employing the case results and rationale of the method construction to qualitatively reflect upon the solution artifact requirements. This showed that the method is able to fulfill the structural requirements such as the combination of BI reference architecture and development process, the effective use of selection criteria make suitable technological selections, as well as the explicit connection from goals and requirements, to functionality and to technology. However, the discussion of the overall approach in Chapter 6 highlighted that, although illustrative scenarios might be employed to illustrate broad aspects of an artifact, an empirical evaluation method is warranted to provide increased validity for the method and to test it in practical contexts.

In summary, it can be said that the research questions formulated in the beginning have been answered. Further, the GOBIA method can be considered a suitable answer to these questions, which, however, lacks further empirical validation before its utility can be generalized beyond the evaluation in this work. Hence, the results lead to several new research opportunities to be tackled in the future, which are detailed in Section 7.2.

7.2 Outlook

Based on the results of this work, three areas of future research opportunities can be identified.

Empirical evaluation of the method. As the GOBIA method intends to solve a practical problem of technology selection and BI architecture formulation, the use of empirical evaluation methods could improve its validity and give a better indication of its utility in empirical settings. For instance, a laboratory setting could explore how business users employ the method. Another scenario could include a detailed practical case study, where the method is applied in a real-world scenario. In such a setting, behavior of the method's users and usage of the method could be examined. Both approaches are advisable to evaluate methods [PRTV12, HMPR04]. This could contribute several findings. For example, certain aspects of the method (e. g., the specific design of process models) could have a positive or negative influence on its utility. Such factors have not been under detailed consideration in this work.

Automation of the method. The contributed method could be implemented as an actual information system, which automates it completely or partially. As Petri nets enable simulation and information flows, they are generally applicable for software implementations. The aspect that tokens in GOBIA.DEV could resemble actual data objects which are manipulated and which was briefly mentioned in Chapter 4, could be instantiated in a prototype. An automation could unlock opportunities to replace manual parts of GOBIA.DEV. For example, the mapping of requirements to BI functionalities in GOBIA.DEV Phase I relies on capabilities of the person conducting the mapping, although a helper table is provided. Here, modern approaches for NLP could help to extract requirements from natural language inputs and suggest suitable BI functionality mappings.

Extended evaluation of BI architecture alternatives. The GOBIA method outputs a set of suitable, customized BI architecture alternatives. These are subject to evaluation in the post-GOBIA phase using factors from the TORE framework, which are briefly outlined. However, the evaluation itself is not part of the GOBIA method. If a contribution to this evaluation and software selection process can be made, these activities could become part of an updated GOBIA method. In particular, quantifiable factors such as through-

put and latency could be measured by means of an *architecture benchmark*. Such an instrument could provide specific figures with respect to measurable technological aspects. While benchmarks for singular technologies such as DWH exist, benchmarks that incorporate novel or a combination of several technologies could be useful for evaluating BI architecture alternatives. With more quantifiable criteria, cost- or score-based evaluations could automatically consider trade-offs, e. g., balance incurred monetary costs and delivered throughput according to weightings provided by a user.

Bibliography

[AB99] M.Jill Austin and Linda D. Brown. Internet Plagiarism: Developing Strategies to Curb Student Academic Dishonesty. *The Internet and Higher Education*, 2(1):21–33, October 1999.

[Aba08] Daniel J. Abadi. *Query execution in column-oriented database systems*. Doctoral dissertation, Massachusetts Institute of Technology Cambridge, MA, USA, 2008.

[ABB+13] Tyler Akidu, Alex Balikov, Kaya Bekiroglu, Slava Chernyak, Josh Haberman, Reuven Lax, Sam McVeety, Daniel Mills, Paul Nordstrom, Sam Whittle, Tyler Akidau, and K Bekiroğlu. MillWheel: Fault-Tolerant Stream Processing at Internet Scale. In *Proceedings of the the VLDB Endowment*, volume 6, pages 734–746, 2013.

[AGG09] Samuil Angelov, Paul Grefen, and Danny Greefhorst. A classification of software reference architectures: Analyzing their success and effectiveness. In *2009 Joint Working IEEE/IFIP Conference on Software Architecture and European Conference on Software Architecture, WICSA/ECSA 2009*, pages 141–150, Cambridge, UK, 2009.

[AGG12] Samuil Angelov, Paul Grefen, and Danny Greefhorst. A framework for analysis and design of software reference architectures. *Information and Software Technology*, 54(4):417–431, April 2012.

[Agg15] Charu C. Aggarwal. *Data Mining: The Textbook*. Springer International Publishing, 2015.

[Agn14] Vijay Srinivas Agneeswaran. Why Look Beyond Hadoop MapReduce? In *Big Data Analytics Beyond Hadoop*, chapter 1, pages 1–17. Pearson Education, Upper Saddle River, New Jersey, USA, 2014.

[AIS93] Rakesh Agrawal, Tomasz Imielinski, and Arun Swami. Mining Association in Large Databases. In *Proceedings of the 1993 ACM SIGMOD international conference on Management of data - SIGMOD '93*, pages 207–216, Washington, D.C., USA, 1993.

[Ama13] Xavier Amatriain. Big & personal: data and models behind Netflix recommendations. In *Proceedings of the 2nd International Workshop on Big Data, Streams and Heterogeneous Source Mining Algorithms, Systems, Programming Models and Applications - BigMine '13*, pages 1–6, New York, New York, USA, 2013. ACM Press.

[Amd67] Gene M. Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the April 18-20, 1967, spring joint computer conference on - AFIPS '67 (Spring)*, page 483, New York, New York, USA, 1967. ACM Press.

[Amo14] Kathleen A. Amos. The ethics of scholarly publishing: exploring differences in plagiarism and duplicate publication across nations. *Journal of the Medical Library Association : JMLA*, 102(2):87–91, April 2014.

[ANS75] ANSI/X3/SPARC Study Group. Interim Report: ANSI/X3/SPARC Study Group on Data Base Management Systems 75-02-08. *Bulletin of ACM SIGMOD*, 7(2):1–140, 1975.

[APA+17] Mehdi Allahyari, Seyedamin Pouriyeh, Mehdi Assefi, Saied Safaei, Elizabeth D. Trippe, Juan B. Gutierrez, and Krys Kochut. A Brief Survey of Text Mining: Classification, Clustering and Extraction Techniques. July 2017.

[Aug05] Sanjiv Augustine. *Managing Agile Projects*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2005.

[BCK03] Len Bass, Paul Clements, and Rick Kazman. *Software Architectures in Practice*. Addison Wesley, Boston, MA, USA, 2nd edition, 2003.

[Bel61] Richard E Bellman. *Adaptive Control Processes: A Guided Tour*. MIT Press, 1961.

[BF16] Alexander Bock and Ulrich Frank. MEMO GoalML: A context-enriched modeling language to support reflective organizational goal planning and decision processes. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 9974 LNCS, pages 515–529. 2016.

[BGL+12] Michael Busch, Krishna Gade, Brian Larson, Patrick Lok, Samuel Luckenbill, and Jimmy Lin. Earlybird: Real-time search at Twitter. *Proceedings - International Conference on Data Engineering*, pages 1360–1369, 2012.

[BK08] Henning Baars and Hans-George Kemper. Management Support with Structured and Unstructured Data—An Integrated Business Intelligence Framework. *Information Systems Management*, 25(2):132–148, March 2008.

[BLT12] Chuck Bear, Andrew Lamb, and Nga Tran. The vertica database: SQL RDBMS for managing big data. In *Proceedings of the 2012 workshop on Management of big data systems - MBDS '12*, page 37, New York, New York, USA, 2012. ACM Press.

[BLW96] Sjaak Brinkkemper, Kalle Lyytinen, and Richard J. Welke, editors. *Method Engineering*. Springer US, Boston, MA, 1996.

[BMH06] Peter F Brown, Rebekah Metz, and Booz Allen Hamilton. Reference Model for Service Oriented Architecture 1.0. Technical Report October, OASIS, 2006.

[BN09] Philip A Bernstein and Eric Newcomer. *Principles of Transaction Processing*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2nd edition, 2009.

[Bre00] Eric A. Brewer. Towards robust distributed systems (abstract). In *Proceedings of the nineteenth annual ACM symposium on Principles of distributed computing - PODC '00*, page 7, New York, New York, USA, 2000. ACM Press.

[Bri96] Sjaak Brinkkemper. Method engineering: Engineering of information systems development methods and tools. *Information and Software Technology*, 38(4 SPEC. ISS.):275–280, 1996.

[BRS95] Jörg Becker, Michael Rosemann, and Reinhard Schütte. Grundsätze ordnungsmäßiger Modellierung. *Wirtschaftsinformatik*, 37(5):435–445, 1995.

[CA13] Carina Cundius and Rainer Alt. Real-Time or Near Real-Time? - Towards a Real-Time Assessment Model. In *Proceedings of the International Conference on Information Systems, (ICIS) 2013, December 15-18, 2013*, Milano, Italy, 2013.

[Cat11] Rick Cattell. Scalable SQL and NoSQL data stores. *ACM SIGMOD Record*, 39(4), 2011.

[CCS12] Hsinchun Chen, Roger H. L. Chiang, and Veda C. Storey. Business Intelligence and Analytics: From Big Data to Big Impact. *MIS Quarterly*, 36(4):1165–1188, 2012.

[CD97] Surajit Chaudhuri and Umeshwar Dayal. An overview of data warehousing and OLAP technology. *ACM SIGMOD Record*, 26(1):65–74, 1997.

[CDE+13] James C Corbett, Jeffrey Dean, Michael Epstein, Andrew Fikes, Christopher Frost, J J Furman, Sanjay Ghemawat, Andrey Gubarev, Christopher Heiser, Peter Hochschild, Wilson Hsieh, Sebastian Kanthak, Eugene Kogan, Hongyi Li, Alexander Lloyd, Sergey Melnik, David Mwaura, David Nagle, Sean Quinlan, Rajesh Rao, Lindsay Rolig, Yasushi Saito, Michal Szymaniak, Christopher Taylor, Ruth Wang, and Dale Woodford. Spanner: Google's globally distributed database. *Google's Globally Distributed Database*, 31(3):1–22, 2013.

[CDG+08] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah a. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber. Bigtable: A Distributed Storage System for Structured Data, 2008.

[CEH+15] Paris Carbone, Stephan Ewen, Seif Haridi, Asterios Katsifodimos, Volker Markl, and Kostas Tzoumas. Apache Flink: Unified Stream and Batch Processing in a Single Engine. *Data Engineering*, 36:28–38, 2015.

[Che76] Peter Pin-Shan Chen. The entity-relationship model—toward a unified view of data. *ACM Transactions on Database Systems*, 1(1):9–36, March 1976.

[Cis17] Cisco. The Zettabyte Era: Trends and Analysis. Technical report, 2017.

[CM12] Gianpaolo Cugola and Alessandro Margara. Processing flows of information. *ACM Computing Surveys*, 44(3):1–62, 2012.

[CML14] Min Chen, Shiwen Mao, and Yunhao Liu. Big Data: A Survey. *Mobile Networks and Applications*, 19(2):171–209, April 2014.

[CMPF13] Carlos E Cuesta, Miguel A Martínez-Prieto, and Javier D Fernández. Towards an Architecture for Managing Big Semantic Data in Real-time. In *Proceedings of the 7th European Conference on Software Architecture*, ECSA'13, pages 45–53, Berlin, Heidelberg, 2013. Springer-Verlag.

[Cod70] E. F. Codd. A Relational Model of Data for Large Shared Data Banks. *Communications of the ACM*, 13(6):377–387, 1970.

[Col11] Ken W Collier. *Agile Analytics: A Value-Driven Approach to Business Intelligence and Data Warehousing*. Addison-Wesley Professional, 1st edition, 2011.

[Cor05] J W Cortada. *The Digital Hand: Volume II: How Computers Changed the Work of American Financial, Telecommunications, Media, and Entertainment Industries*. Oxford University Press, 2005.

[CTT+14] Ugur Cetintemel, Nesime Tatbul, Kristin Tufte, Hao Wang, Stanley Zdonik, Jiang Du, Tim Kraska, Samuel Madden, David Maier, John Meehan, Andrew Pavlo, Michael Stonebraker, and Erik Sutherland. S-Store: A Streaming NewSQL System for Big Velocity Applications. *Proceedings of the VLDB Endowment*, 7(13):1633–1636, August 2014.

[Cur16] Edward Curry. *New Horizons for a Data-Driven Economy*. Springer International Publishing, Cham, 2016.

[CZ14] C.L. Philip Chen and Chun-Yang Zhang. Data-intensive applications, challenges, techniques and technologies: A survey on Big Data. *Information Sciences*, 275:314–347, August 2014.

[dB16] Anatole de Baudot. *L'architecture. Le passé.–Le présent.* Laurens, Henri, 1916.

[De 11] Dean De Cock. Ames, Iowa: Alternative to the Boston Housing Data as an End of Semester Regression Project. *Journal of Statistics Education*, 19(3), 2011.

[Dem86] W Edwards Deming. *Out of the crisis / W. Edwards Deming*. Massachusetts Institute of Technology, Center for Advanced Engineering Study ; Cambridge University Press Cambridge, Mass. : Cambridge, Cambridge, MA, 1986.

[Dev65] R M Devens. *Cyclopædia of Commercial and Business Anecdotes: Comprising Interesting Reminiscences and Facts, Remarkable Traits and Humors ... of Merchants, Traders, Bankers ... Etc. in All Ages and Countries*. Number Vol. 1 in Cyclopædia of Commercial and Business Anecdotes. D. Appleton, New York, NY, USA, 1865.

[DG04] Jeffrey Dean and Sanjay Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. In *Proceedings of the 6th Conference on Symposium on Operating Systems Design & Implementation - Volume 6*, OSDI'04, page 10, Berkeley, CA, USA, 2004. USENIX Association.

[DGL12] Lisette De Vries, Sonja Gensler, and Peter S.H. Leeflang. Popularity of Brand Posts on Brand Fan Pages: An Investigation of the Effects of Social Media Marketing. *Journal of Interactive Marketing*, 26(2):83–91, 2012.

[DHJ+07] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. Dynamo: amazon's highly available key-value store. In *Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles - SOSP '07*, page 205, 2007.

[Die12] Francis X Diebold. A Personal Perspective on the Origin(s) and Development of "Big Data": The Phenomenon, the Term, and the Discipline. Technical report, University of Pennsylvania, Philadelphia, USA, 2012.

[DMS06] David J. DeWitt, Samuel Madden, and Michael Stonebraker. How to Build a High-Performance Data Warehouse, 2006.

[DN10] Uwe Draisbach and Felix Naumann. DuDe: The duplicate detection toolkit. In *Proceedings of the International Workshop on Quality in Databases (QDB)*, Singapore, Malaysia, 2010.

[DP97] Thomas H Davenport and Laurence Prusak. *Information Ecology: Mastering the Information and Knowledge Environment*. Oxford University Press, 1st edition, 1997.

[DT15] Maxwell Dayvson Da Silva and Hugo Lopes Tavares. *Redis Essentials*. Packt Publishing, 2015.

[DvLF93] Anne Dardenne, Axel van Lamsweerde, and Stephen Fickas. Goal-directed requirements acquisition. *Science of Computer Programming*, 20(1-2):3–50, April 1993.

[EK08] Khaled El Emam and A. Günes Koru. A Replicated Survey of IT Software Project Failures. *IEEE Software*, 25(5):84–90, September 2008.

[Ell14] Byron Ellis. *Real-Time Analytics: Techniques to Analyze and Visualize Streaming Data*. Wiley Publishing, 1st edition, 2014.

[EN16] Ramez Elmasri and Shamkant B Navathe. *Fundamentals of Database Systems*. Pearson, Boston, MA, USA, 7th edition, 2016.

[FCP+12] Franz Färber, Sang Kyun Cha, Jürgen Primsch, Christof Bornhövd, Stefan Sigg, and Wolfgang Lehner. SAP HANA Database: Data Management for Modern Business Applications. *SIGMOD Rec.*, 40(4):45–51, 2012.

[Fel13] Ronen Feldman. Techniques and applications for sentiment analysis. *Communications of the ACM*, 56(4):82, April 2013.

[FG10] Randy Farmer and Bryce Glass. *Building Web Reputation Systems*. Yahoo! Press, USA, 1st edition, 2010.

[Fin98] John Patrick Finnegan. *Military intelligence*. Center of Military History, United States Army, Washington, D.C., army lineage series edition, 1998.

[FM91] Kevin Forsberg and Harold Mooz. The Relationship of System Engineering to the Project Cycle. *INCOSE International Symposium*, 1(1):57–65, October 1991.

[FML+12] Franz Färber, Norman May, Wolfgang Lehner, Philipp Große, Ingo Müller, Hannes Rauhe, and Jonathan Dees. The SAP HANA Database – An Architecture Overview. *IEEE Data Eng. Bull.*, 35(1):28–33, 2012.

[FN86] L Fahey and V K Narayanan. *Macroenvironmental Analysis for Strategic Management*. West series in strategic management. West, 1986.

[Fra12] Bill Franks, editor. *Taming the Big Data Tidal Wave*. John Wiley & Sons, Inc., Hoboken, NJ, USA, January 2012.

[FSC12] F. C. Fang, R. G. Steen, and A. Casadevall. Misconduct accounts for the majority of retracted scientific publications. *Proceedings of the National Academy of Sciences*, 109(42):17028–17033, October 2012.

[FSF+14] Ulrich Frank, Stefan Strecker, Peter Fettke, Jan vom Brocke, Jörg Becker, and Elmar Sinz. The Research Field "Modeling Business Information Systems". *Business & Information Systems Engineering*, 6(1):39–43, 2014.

[Gal15] Matthias Galster. Software Reference Architectures: Related Architectural Concepts and Challenges. In *Proceedings of the 1st International Workshop on Exploring Component-based Techniques for Constructing Reference Architectures - CobRA '15*, pages 5–8, 2015.

[GDK18] Evangelos Gkolemis, Katerina Doka, and Nectarios Koziris. Automatic Scaling of Resources in a Storm Topology. In Dan Alistarh, Alex Delis, and George Pallis, editors, *Algorithmic Aspects of Cloud Computing*, pages 157–169, Cham, 2018. Springer International Publishing.

[Gee13] Bas Geerdink. A Reference Architecture for Big Data Solutions. *The 8th International Conference for Internet Technology and Secured Transactions*, pages 71–76, 2013.

[GGL03] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The Google file system. In *Proceedings of the nineteenth ACM symposium on Operating systems principles - SOSP '03*, page 29, New York, New York, USA, 2003. ACM Press.

[GH03] E Grochowski and RD Halem. Technological impact of magnetic hard disk drives on storage systems. *IBM Systems Journal*, 42(2):338–346, 2003.

[GHJV94] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-oriented Software*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1994.

[GHP14] G. George, M. R. Haas, and A. Pentland. Big Data and Management. *Academy of Management Journal*, 57(2):321–326, April 2014.

[GJ90] Peter M. Ginter and W. Jack Duncan. Macroenvironmental analysis for strategic management. *Long Range Planning*, 23(6):91–100, 1990.

[GKV06] Danny Greefhorst, Henk Koning, and Hans Van Vliet. The many faces of architectural descriptions. *Information Systems Frontiers*, 8(2):103–113, 2006.

[GL02] Seth Gilbert and Nancy Lynch. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *ACM SIGACT News*, 33(2):51–59, June 2002.

[GMS92] Hector Garcia-Molina and Kenneth Salem. Main memory database systems: An overview. *IEEE Transactions on Knowledge and Data Engineering*, 4(6):509–516, 1992.

[GR11] John Gantz and David Reinsel. Extracting Value from Chaos State of the Universe: An Executive Summary. *IDC iView*, (June):1–12, 2011.

[GRH+13] Ahmad Ghazal, Tilmann Rabl, Minqing Hu, Francois Raab, Meikel Poess, Alain Crolotte, and Hans-Arno Jacobsen. BigBench: towards an industry standard benchmark for big data analytics. In *Proceedings of the 2013 international conference on Management of data - SIGMOD '13*, page 1197, New York, New York, USA, 2013. ACM Press.

[GRM15] Wilfried Grossmann and Stefanie Rinderle-Ma. *Fundamentals of Business Intelligence.* Data-Centric Systems and Applications. Springer Berlin Heidelberg, Berlin, Heidelberg, 2015.

[HAC+16] Jennifer Horkoff, Fatma Basak Aydemir, Evellin Cardoso, Tong Li, Alejandro Mate, Elda Paja, Mattia Salnitri, John Mylopoulos, and Paolo Giorgini. Goal-Oriented Requirements Engineering: A Systematic Literature Map. In *2016 IEEE 24th International Requirements Engineering Conference (RE)*, pages 106–115. IEEE, September 2016.

[Hen08] Jean-Jacques Pierre Henry. *The Testing Network.* Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[HK06] Samuel B. Hwang and Sungho Kim. Dynamic Pricing Algorithm for E-Commerce. In *Advances in Systems, Computing Sciences and Software Engineering*, pages 149–155. Springer Netherlands, Dordrecht, 2006.

[HKP11] Jiawei Han, Micheline Kamber, and Jian Pei. *Data Mining: Concepts and Techniques.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 3rd edition, 2011.

[HMPR04] Alan R Hevner, Salvatore T March, Jinsoo Park, and Sudha Ram. Design Science in Information Research. *MIS Quarterly*, 28(1):75–105, 2004.

[Hof15] S Hoffman. *Apache Flume: Distributed Log Collection for Hadoop - Second Edition.* Community Experience Distilled. Packt Publishing, 2nd edition, 2015.

[Hor00] A S Hornby. *Oxford Advanced Learner's Dictionary.* Oxford University Press, Oxford, 6th edition, 2000.

[HPS+16] Peter Heller, Dee Piziak, Robert Stackowiak, Art Licht, Tom Luckenbach, Bob Cuathen, Misra Avishkar, John Wyant, and Jeff Knudsen. An Enterprise Architect ' s Guide to Big Data. Technical Report March, Oracle Corportation, 2016.

[HWCL14] Han Hu, Yonggang Wen, Tat-Seng Chua, and Xuelong Li. Toward Scalable Systems for Big Data Analytics: A Technology Tutorial. *IEEE Access*, 2:652–687, 2014.

[IBM13] IBM Software. Descriptive, predictive, prescriptive: Transforming asset and facilities management with analytics. Technical Report October, 2013.

[IDC14] IDC / EMC. The Digital Universe of Opportunities. Technical report, 2014.

[IEE03] IEEE 802.3af-2003 - IEEEE Standard for Information Technology. Standard, Institute of Electrical and Electronics Engineers, New York, NY, USA, 2003.

[IEE09] IEEE 802.3at-2009 - IEEE Standard for Information technology. Standard, Institute of Electrical and Electronics Engineers, New York, NY, USA, 2009.

[IEE11] IEEE 802.11s-2011 - IEEEE Standard for Information Technology. Standard, Institute of Electrical and Electronics Engineers, New York, NY, USA, 2011.

[Inm96] William Inmon. *Building the Data Warehouse*. John Wiley & Sons Inc., New York, New York, USA, 2nd edition, 1996.

[Inm05] William Inmon. *Building the Data Warehouse*. Wiley, Burlington, MA, 4th edition, 2005.

[Inm16] William Inmon. *Data Lake Architecture: Designing the Data Lake and Avoiding the Garbage Dump*. Technics Publications, Basking Ridge, NJ, USA, 1 edition, 2016.

[ISN08] William H Inmon, Derek Strauss, and Genia Neushloss. *DW 2.0: The Architecture for the Next Generation of Data Warehousing*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2008.

[ISO87] ISO 9075:1987 - Information processing systems – Database language – SQL. Standard, International Organization for Standardization, Geneva, Switzerland, 1987.

[ISO92] ISO/IEC 9075:1992 - Information technology – Database languages – SQL. Standard, International Organization for Standardization, Geneva, Switzerland, 1992.

[ISO99] ISO/IEC 9075-1:1999 - Information technology – Database languages – SQL – Part 1: Framework (SQL/Framework). Standard, International Organization for Standardization, Geneva, Switzerland, 1999.

[ISO03] ISO/IEC 9075-1:2013 - Information technology – Database languages – SQL – Part 1: Framework (SQL/Framework). Standard, International Organization for Standardization, Geneva, Switzerland, 2003.

[ISO10] ISO/IEC/IEEE 24765:2010 - Systems and software engineering – Vocabulary. Standard, International Organization for Standardization, Geneva, Switzerland, 2010.

[ISO11] ISO/IEC/IEEE 42010:2011 - Systems and software engineering – Architecture description. Standard, International Organization for Standardization, Geneva, Switzerland, 2011.

[ISO13] ISO/IEC ISO/IEC 19510:2013 - Information technology – Object Management Group Business Process Model and Notation. Standard, International Organization for Standardization, Geneva, Switzerland, 2013.

[ISO16] ISO 9075-1:2016 - Information technology – Database languages – SQL – Part 1: Framework (SQL/Framework). Standard, International Organization for Standardization, Geneva, Switzerland, 2016.

[Jac09] Adam Jacobs. The Pathologies of Big Data. *Communications of the ACM*, 52(8):36–44, 2009.

[JS12] Andrea Janes and Giancarlo Succi. The dark side of agile software development. *Proceedings of the ACM international symposium on New ideas, new paradigms, and reflections on programming and software*, pages 215–227, 2012.

[JSNJ15] Nenad Jukić, Abhishek Sharma, Svetlozar Nestorov, and Boris Jukić. Augmenting Data Warehouses with Big Data. *Information Systems Management*, 32(3):200–209, 2015.

[KBF+15] Sanjeev Kulkarni, Nikunj Bhagat, Masong Fu, Vikas Kedigehalli, Christopher Kellogg, Sailesh Mittal, Jignesh M. Patel, Karthik Ramasamy, and Siddarth Taneja. Twitter Heron. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data - SIGMOD '15*, pages 239–250, 2015.

[KBM10] Hans-Georg Kemper, Henning Baars, and Walid Mehanna. *Business Intelligence – Grundlagen und praktische Anwendungen.* Vieweg+Teubner, Wiesbaden, 2010.

[KC04] Ralph Kimball and Joe Caserta. *The Data Warehouse ETL Toolkit: Practical Techniques for Extracting, Cleaning, Conforming and Delivering Data.* John Wiley & Sons, 2004.

[KG15] S. P. T. Krishnan and Jose L. Ugia Gonzalez. Google Cloud Dataflow. In *Building Your Next Big Thing with Google Cloud Platform*, pages 255–275. Apress, Berkeley, CA, 2015.

[Kim96] Ralph Kimball. *The Data Warehouse Toolkit.* Wiley Publishing, New York, NY, USA, 1st edition, 1996.

[KK15] M Kleppmann and J Kreps. Kafka, Samza and the Unix Philosophy of Distributed Data. *IEEE Data Engineering Bulletin*, pages 1–11, 2015.

[KKH08] Kevin E Kline, Daniel Kline, and Brand Hunt. *SQL in a nutshell - a desktop quick reference.* O'Reilly, Sebastopol, CA, 3rd edition, 2008.

[KL15] S. Kaavya and G. G. Lakshmipriya. Multimedia Indexing and Retrieval: Recent research work and their challenges. *2015 3rd International Conference on Signal Processing, Communication and Networking, ICSCN 2015*, 2015.

[KLS95] John Krogstie, Odd Ivar Lindland, and Guttorm Sindre. Defining Quality Aspects for Conceptual Models. In *Proceedings of the IFIP International Working Conference on Information System Concepts: Towards a Consolidation of Views*, pages 216–231, London, UK, UK, 1995. Chapman & Hall, Ltd.

[KNT06] R Kumar, J Novak, and A Tomkins. Structure and evolution of online social networks. In *Proceedings of 12th International Conference on Knowledge Discovery in Data Mining*, pages 611 – 617, 2006.

[KR13] Ralph Kimball and Margy Ross. *The Data Warehouse Toolkit: The Definitive Guide to Dimensional Modeling.* Wiley Publishing, 3rd edition, 2013.

[KWM11] Efthymios Kouloumpis, Theresa Wilson, and Johanna D Moore. Twitter Sentiment Analysis: The Good the Bad and the OMG! In Lada A Adamic, Ricardo A Baeza-Yates, and Scott Counts, editors, *ICWSM*. The AAAI Press, 2011.

[Lan01] Doug Laney. 3D Data Management: Controlling Data Volume, Velocity, and Variety. Technical report, META Group, February 2001.

[LDDD91] A. Lamsweerde, A. Dardenne, B. Delcourt, and F. Dubisy. The KAOS Project: Knowledge Acquisition in Automated Specification of Software. *Proc. AAAI Spring Symp. Series*, pages 59–62, 1991.

[Leh03] Wolfgang Lehner. *Datenbanktechnologie für Data-Warehouse-Systeme*. d.punkt Verlag, Heidelberg, 2003.

[LFV+12] Andrew Lamb, Matt Fuller, Ramakrishna Varadarajan, Nga Tran, Ben Vandiver, Lyric Doshi, and Chuck Bear. The vertica analytic database: C-Store 7 Years Later. *Proceedings of the VLDB Endowment*, 5(12):1790–1801, August 2012.

[Liu12] Bing Liu. Sentiment Analysis and Opinion Mining. *Synthesis Lectures on Human Language Technologies*, 5(1):1–167, May 2012.

[LKRH15] Sara Landset, Taghi M Khoshgoftaar, Aaron N Richter, and Tawfiq Hasanin. A survey of open source tools for machine learning with big data in the Hadoop ecosystem. *Journal of Big Data*, 2(1):24, 2015.

[LL15] Daniel T Larose and Chantal D Larose. *Data Mining and Predictive Analytics*. Wiley Publishing, 2nd edition, 2015.

[LLD16] Martin Andreoni Lopez, Antonio Gonzalez Pastana Lobato, and Otto Carlos M. B. Duarte. A Performance Comparison of Open-Source Stream Processing Platforms. In *2016 IEEE Global Communications Conference (GLOBECOM)*, pages 1–6. IEEE, December 2016.

[LLL+12] George Lee, Jimmy Lin, Chuang Liu, Andrew Lorek, and Dmitriy Ryaboy. The unified logging infrastructure for data analytics at Twitter. *Proceedings of the VLDB Endowment*, 5(12):1771–1780, August 2012.

[LRU14] Jure Leskovec, Anand Rajaraman, and Jeffrey D. Ullman. *Mining of Massive Datasets*. Palo Alto, CA, 2014.

[LS10] Wolfgang Lehner and Kai-Uwe Sattler. Database as a service (DBaaS). In *2010 IEEE 26th International Conference on Data Engineering (ICDE 2010)*, pages 1216–1217. IEEE, 2010.

[LSS94] Odd Ivar Lindland, Guttorm Sindre, and Arne Sølvberg. Understanding Quality in Conceptual Modeling. *IEEE Software*, 11(2):42–49, 1994.

[Luh58] H. P. Luhn. A Business Intelligence System. *IBM Journal of Research and Development*, 2(4):314–319, October 1958.

[LWL+17] Weibo Liu, Zidong Wang, Xiaohui Liu, Nianyin Zeng, Yurong Liu, and Fuad E. Alsaadi. A survey of deep neural network architectures and their applications. *Neurocomputing*, 234:11–26, 2017.

[MAMASF13] Dana Movshovitz-Attias, Yair Movshovitz-Attias, Peter Steenkiste, and Christos Faloutsos. Analysis of the reputation system and user contributions on a question answering website. In *Proceedings of the 2013 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining - ASONAM '13*, pages 886–893, 2013.

[Mar03] Robert Cecil Martin. *Agile Software Development: Principles, Patterns, and Practices.* Prentice Hall PTR, Upper Saddle River, NJ, USA, 2003.

[Mar15] Bernard Marr. *Big Data: Using SMART Big Data, Analytics and Metrics To Make Better Decisions and Improve Performance.* Wiley, 1st edition, 2015.

[Mar16] Bernard Marr. *Big Data in Practice: How 45 Successful Companies Used Big Data Analytics to Deliver Extraordinary Results.* John Wiley & Sons, Ltd, Chichester, UK, April 2016.

[MB15] Gianmarco De Francisci Morales and Albert Bifet. SAMOA: Scalable Advanced Massive Online Analysis. *Journal of Machine Learning Research*, 16:149–153, 2015.

[MCB+11] J Manyika, M Chui, B Brown, J Bughin, R Dobbs, C Roxburgh, and Angela Hung Byers. Big data: The next frontier for innovation, competition and productivity. Technical Report June, 2011.

[McC96] Steve McConnell. *Rapid Development: Taming Wild Software Schedules.* Microsoft Press, Redmond, WA, USA, 1st edition, 1996.

[MCF+95] Richard J. Mayer, John W. Crump, Ronald Fernandes, Arthur Keen, and Michael K. Painter. Information Integration for Concurrent Engineering (IICE) Compendum of Methods Report. Technical Report JUNE, Air Force Materiel Command, Wright-Patterson Air Force Base, Ohio, 1995.

[McG15] John McGee. Market-Based View. In *Wiley Encyclopedia of Management*, pages 1–1. John Wiley & Sons, Ltd, Chichester, UK, January 2015.

[MDL+13] Gilad Mishne, Jeff Dalton, Zhenghua Li, Aneesh Sharma, and Jimmy Lin. Fast data in the era of big data: Twitter's real-time related query suggestion architecture. In *Proceedings of the 2013 international conference on Management of data - SIGMOD '13*, page 1147, New York, New York, USA, 2013. ACM Press.

[MF04] Roger Macnicol and Blaine French. Sybase IQ Multiplex Designed For Analytics. In *Proceedings of the Thirtieth international conference on Very large data bases*, pages 1227–1230, Toronto, Canada, 2004. VLDB Endowment.

[MG11] P M Mell and T Grance. The NIST definition of cloud computing. Technical report, National Institute of Standards and Technology, Gaithersburg, MD, 2011.

[ML16] Cedrine Madera and Anne Laurent. The next information architecture evolution. In *Proceedings of the 8th International Conference on Management of Digital EcoSystems - MEDES*, pages 174–180, New York, New York, USA, 2016. ACM Press.

[MN10] Ronald D Moen and Clifford L Norman. Circling Back: Clearing up the myths about the Deming cycle and seeing how it keeps evolving. *Quality Progress*, 1(November):22–28, November 2010.

[Moe09] Ronald Moen. Foundation and History of the PDSA Cycle. Technical report, 2009.

[Mon13] Gregory Mone. Beyond Hadoop. *Communications of the ACM*, 56(1):22, 2013.

[MP92] Joseph T. Mahoney and J. Rajendran Pandian. The resource-based view within the conversation of strategic management. *Strategic Management Journal*, 13(5):363–380, June 1992.

[MPCAF15] Miguel A. Martínez-Prieto, Carlos E. Cuesta, Mario Arias, and Javier D. Fernández. The Solid architecture for real-time management of big semantic data. *Future Generation Computer Systems*, 47:62–79, 2015.

[MRvdA10] J Mendling, H A Reijers, and W M P van der Aalst. Seven Process Modeling Guidelines (7PMG). *Information and Software Technology*, 52(2):127–136, 2010.

[MS95] Salvatore T. March and Gerald F. Smith. Design and natural science research on information technology. *Decision Support Systems*, 15(4):251–266, 1995.

[MS12] Donald Miner and Adam Shook. *MapReduce Design Patterns: Building Effective Algorithms and Analytics for Hadoop and Other Systems*. O'Reilly Media, Inc., 1st edition, 2012.

[MSdB14] Arinto Murdopo, Antonio Severien, Ginamarco de Francisci Morales, and Albert Bifet. Apache Samoa - Developers Guide. Technical report, Yahoo Labs Barcelona, Barcelona, 2014.

[MSM18] Grégoire Montavon, Wojciech Samek, and Klaus-Robert Müller. Methods for interpreting and understanding deep neural networks. *Digital Signal Processing*, 73:1–15, February 2018.

[MvdL10] Gerrit Muller and Pierre van de Laar. Researching Reference Architectures and Their Relationship with Frameworks, Methods, Techniques, and Tools. *Insight*, 13(2):24–30, 2010.

[MW15] Nathan Marz and James Warren. *Big Data, Principles and best practices of scalable real-time data systems*, volume 37. Manning Publications Co., Shelter Island, NY, USA, 2015.

[Nas16] Muhammad Anis Uddin Nasir. Fault Tolerance for Stream Processing Engines. *arXiv:1605.00928 [cs]*, pages 1–10, 2016.

[NB13] Paul Newton and Helen Bristoll. *PESTLE Analysis: Strategy Skills*. 2013.

[Nee15] Nishant Neeraj. *Mastering Apache Cassandra*. Packt Publishing, 2nd edition, 2015.

[Nev11] Benjamin Nevarez. *Inside the SQL Server Query Optimizer*. Red Gate books, 2011.

[NHC07] Rajiv Nag, Donald C. Hambrick, and Ming-Jer Chen. What is strategic management, really? Inductive derivation of a consensus definition of the field. *Strategic Management Journal*, 28(9):935–955, September 2007.

[NIS15a] NIST Big Data Public Working Group. NIST Big Data Interoperability Framework: Volume 5, Architectures White Paper Survey. Technical report, National Institute of Standards and Technology, Gaithersburg, MD, October 2015.

[NIS15b] NIST Big Data Public Working Group. NIST Big Data Interoperability Framework: Volume 6, Reference Architecture. Technical report, National Institute of Standards and Technology, Gaithersburg, MD, October 2015.

[NIS15c] NIST Big Data Public Working Group. NIST Special Publication 1500-1 - NIST Big Data Interoperability Framework: Volume 1, Definitions. Technical report, National Institute of Standards and Technology, 2015.

[NMK15] Thomas Neumann, Tobias Mühlbauer, and Alfons Kemper. Fast Serializable Multi-Version Concurrency Control for Main-Memory Database Systems. *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data - SIGMOD '15*, pages 677–689, 2015.

[NW11] Saggi Nevo and Michael Wade. Firm-level benefits of IT-enabled resources: A conceptual extension and an empirical assessment. *The Journal of Strategic Information Systems*, 20(4):403–418, December 2011.

[O'B39] John G O'Brien. *Wholesale Business Intelligence and Southern and Western Merchants' Pocket Directory to the Principal Mercantile Houses in the City of Philadelphia, for the Year 1839*. O'Brien, Philadelphia, USA, 1839.

[ÖBF+11] Hubert Österle, Jörg Becker, Ulrich Frank, Thomas Hess, Dimitris Karagiannis, Helmut Krcmar, Peter Loos, Peter Mertens, Andreas Oberweis, and Elmar J. Sinz. Memorandum on design-oriented information systems research. *European Journal of Information Systems*, 20(1):7–10, 2011.

[OBRS10] Brendan O'Connor, Ramnath Balasubramanyan, Bryan R Routledge, and Noah A Smith. From Tweets to Polls: Linking Text Sentiment to Public Opinion Time Series. In William W Cohen and Samuel Gosling, editors, *ICWSM*. The AAAI Press, 2010.

[Ora14] Oracle Corporation. Information Management and Big Data - A Reference Architecture | Oracle White Paper. Technical report, Oracle Corporation, Redwood Shores, CA, USA, 2014.

[ORS+08] Christopher Olston, Benjamin Reed, Utkarsh Srivastava, Ravi Kumar, and Andrew Tomkins. Pig latin: a not-so-foreign language for data processing. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data - SIGMOD '08*, page 1099, New York, New York, USA, 2008. ACM Press.

[PA16] Andrew Pavlo and Matthew Aslett. What's Really New with NewSQL? *ACM SIGMOD Record*, 45(2):45–55, 2016.

[PCN+11] Jonathan D. Power, Alexander L. Cohen, Steven M. Nelson, Gagan S. Wig, Kelly Anne Barnes, Jessica A. Church, Alecia C. Vogel, Timothy O. Laumann, Fran M. Miezin, Bradley L. Schlaggar, and Steven E. Petersen. Functional Network Organization of the Human Brain. *Neuron*, 72(4):665–678, November 2011.

[PCWA15] Nicolas Prat, Isabelle Comyn-Wattiau, and Jacky Akoka. A Taxonomy of Evaluation Methods for Information Systems Artifacts. *Journal of Management Information Systems*, 32(3):229–267, 2015.

[Pet62] Carl Adam Petri. *Kommunikation mit Automaten*. PhD thesis, Universität Hamburg, 1962.

[PF13] Foster Provost and Tom Fawcett. *Data Science for Business: What You Need to Know About Data Mining and Data-analytic Thinking*. O'Reilly Media, Inc., 1st edition, 2013.

[Phi12] Kevin Lane Keller Philip Kotler. *Marketing Management, 14th Edition*. Prentice Hall, 14 edition, 2012.

[PI05] Gabriele Piccoli and Blake Ives. Review: IT-Dependent Strategic Initiatives and Sustained Competitive Advantage: A Review and Synthesis of the Literature. *MIS Quarterly*, 29(4):747–776, 2005.

[PM85] Michael E. Porter and Victor E. Millar. How information gives you competitive advantage. *Harvard business review*, 63(4 (July-August)):149–160, 1985.

[Poh10] Klaus Pohl. *Requirements Engineering: Fundamentals, Principles, and Techniques*. Springer, Heidelberg, 2010.

[Por79] Michael Porter. How Competitive Forces Shape Strategy. *Harvard business Review*, 57(2):137–145, 1979.

[Pow04] Daniel Power. Decision Support Systems: From the Past to the Future. In *10th Americas Conference on Information Systems, AMCIS 2004, New York, NY, USA, August 6-8, 2004*, page 242. Association for Information Systems, 2004.

[PP15] Pekka Pääkkönen and Daniel Pakkala. Reference Architecture and Classification of Technologies, Products and Services for Big Data Systems. *Big Data Research*, 2(4):166–186, 2015.

[PR15] Klaus Pohl and Chris Rupp. *Requirements Engineering Fundamentals*. rockynook, Santa Barbara, CA, USA, 2nd edition, 2015.

[PRTV12] Ken Peffers, Marcus Rothenberger, Tuure Tuunanen, and Reza Vaezi. Design Science Research Evaluation. In Ken Peffers, Marcus Rothenberger, and Bill Kuechler, editors, *Design Science Research in Information Systems. Advances in Theory and Practice*, pages 398–410, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[Psa17] A G Psaltis. *Streaming Data: Designing the Real-Time Pipeline*. Manning Publications Company, Shelter Island, NY, USA, 2017.

[PSE+09] Martin Potthast, Benno Stein, Andreas Eiselt, Alberto Barrón-Cedeño, and Paolo Rosso. Overview of the 1st International Competition on Plagiarism Detection. *Proceedings of the SEPLN'09 Workshop on Uncovering Plagiarism, Authorship and Social Software Misuse (PAN 09)*, pages 1–9, 2009.

[PTRC07] Ken Peffers, Tuure Tuunanen, Marcus A. Rothenberger, and Samir Chatterjee. A Design Science Research Methodology for Information Systems Research. *Journal of Management Information Systems*, 24(3):45–77, December 2007.

[Rap14] RapidMiner. An Introduction Advanced Analytics. Technical report, 2014.

[Rei85] Wolfgang Reisig. *Petri Nets*. Springer Berlin Heidelberg, Berlin, Heidelberg, 1985.

[Rei16] Charles Reiss. Understanding Memory Configurations for In-Memory Analytics. Technical report, University of California at Berkeley, 2016.

[RGS02] Nils Rasmussen, Paul S. Goldy, and Per O. Solli. *Financial Business Intelligence: Trends, Technology, Software Selection, and Implementation*. John Wiley & Sons, 2002.

[RK10] Ariel Rabkin and Randy Katz. Chukwa: A system for reliable large-scale log collection. In *LISA ' 11 : 25th Large Installation System Administration Conference*, pages 163–177, 2010.

[RN09] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall Press, Upper Saddle River, NJ, USA, 3rd edition, 2009.

[Ros03] Michael Rosemann. Preparation of Process Modeling. In Jörg Becker, Martin Kugeler, and Michael Rosemann, editors, *Process Management: A Guide for the Design of Business Processes*, pages 41–78. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003.

[Roy70] Dr. Winston W. Royce. Managing the Development of large Software Systems. *IEEE Wescon*, (August):1–9, 1970.

[Rus11] Philip Russom. Big Data Analytics. Technical report, TDWI Research, 2011.

[RW12] Eric Redmond and Jim R Wilson. *Seven Databases in Seven Weeks: A Guide to Modern Databases and the NoSQL Movement*. Pragmatic Bookshelf, 2012.

[RWE15] Ian Robinson, Jim Webber, and Emil Eifrem. *Graph Databases: New Opportunities for Connected Data*. O'Reilly Media, Inc., 2nd edition, 2015.

[SAB+05] Mike Stonebraker, D.J. Abadi, Adam Batkin, Xuedong Chen, Mitch Cherniack, Miguel Ferreira, Edmond Lau, Amerson Lin, Sam Madden, and E. O'Neil. C-store: a column-oriented DBMS. In *Proceedings of the 31st international conference on Very large data bases*, page 564. VLDB Endowment, 2005.

[SAD+10] Michael Stonebraker, Daniel Abadi, David J. DeWitt, Sam Madden, Erik Paulson, Andrew Pavlo, and Alexander Rasin. MapReduce and parallel DBMSs: Friends or Foes? *Communications of the ACM*, 53(1):64, January 2010.

[SBL02] Josef Schiefer, Robert M Bruckner, and Beate List. A Holistic Approach For Managing Requirements Of Data Warehouse Systems. *Eight Americas Conference on Information Systems*, pages 77–87, 2002.

[SC05] M. Stonebraker and U. Cetintemel. "One Size Fits All": An Idea Whose Time Has Come and Gone. In *21st International Conference on Data Engineering (ICDE'05)*, pages 2–11. IEEE, 2005.

[SCG+17] Eduardo Felipe Zambom Santana, Ana Paula Chaves, Marco Aurelio Gerosa, Fabio Kon, and Dejan S. Milojicic. Software Platforms for Smart Cities. *ACM Computing Surveys*, 50(6):1–37, November 2017.

[Sch02] August-Wilhelm Scheer. *ARIS — Vom Geschäftsprozess zum Anwendungssystem.* Springer Berlin Heidelberg, Berlin, Heidelberg, 2002.

[Sch08] Michael Schrader. Understanding an OLAP Solution from Oracle, 2008.

[Sch15] Jürgen Schmidhuber. Deep Learning in neural networks: An overview. *Neural Networks*, 61:85–117, 2015.

[SÇZ05] Michael Stonebraker, Uğur Çetintemel, and Stan Zdonik. The 8 requirements of real-time stream processing. *ACM SIGMOD Record*, 34(4):42–47, 2005.

[SD14] Seba Susan and Mayank Dwivedi. Dynamic Growth of Hidden-Layer Neurons Using the Non-extensive Entropy. In *2014 Fourth International Conference on Communication Systems and Network Technologies*, pages 491–495. IEEE, April 2014.

[SEC+13] Jeff Shute, Stephan Ellner, John Cieslewicz, Ian Rae, Traian Stancescu, Himani Apte, Radek Vingralek, Bart Samwel, Ben Handy, Chad Whipkey, Eric Rollins, Mircea Oancea, Kyle Littlefield, and David Menestrina. F1: a distributed SQL database that scales. *Proceedings of the VLDB Endowment*, 6(11):1068–1079, 2013.

[SF12] Pramod J Sadalage and Martin Fowler. *NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence.* Addison Wesley, Crawfordsville, Indiana, USA, 2012.

[SG07] Willem E. Saris and Irmtraud N. Gallhofer. *Design, Evaluation, and Analysis of Questionnaires for Survey Research.* John Wiley & Sons, Inc., Hoboken, NJ, USA, June 2007.

[SGG12] Abraham Silberschatz, Peter B Galvin, and Greg Gagne. *Operating System Concepts.* Wiley Publishing, 9th edition, 2012.

[SGR15] Michael Schaarschmidt, Felix Gessert, and Norbert Ritter. Towards Automated Polyglot Persistence. In Thomas Seidl, Norbert Ritter, Harald Schöning, Kai-Uwe Sattler, Theo Härder, Steffen Friedrich, and Wolfram Wingerath, editors, *Datenbanksysteme für Business, Technologie und Web (BTW 2015)*, pages 73–82, Bonn, 2015. Gesellschaft für Informatik e.V.

[SGT98] Albert H. Segars, Varun Grover, and James T.C. Teng. Strategic Information Systems Planning: Planning System Dimensions, Internal Coalignment, and Implications for Planning Effectiveness. *Decision Sciences*, 29(2):303–341, March 1998.

[Shv10] Konstantin V Shvachko. HDFS Scalability: The Limits to Growth. *;login; - The Linux Magazine*, April 2010.

[SKG+12] Roshan Sumbaly, Jay Kreps, Lei Gao, Alex Feinberg, Chinmay Soman, and Sam Shah. Serving Large-scale Batch Computed Data with Project Voldemort. In *Proceedings of the 10th USENIX Conference on File and Storage Technologies*, FAST'12, page 18, Berkeley, CA, USA, 2012. USENIX Association.

[SKS13] Roshan Sumbaly, Jay Kreps, and Sam Shah. The big data ecosystem at LinkedIn. In *Proceedings of the 2013 international conference on Management of data - SIGMOD '13*, page 1125, New York, New York, USA, 2013. ACM Press.

[SM13] Adnan Shaout and Kevin Mcgirr. Real-Time Systems in Automotive Applications: Vehicle Stability Control. *Electrical Engineering Research*, 1(4):83–95, 2013.

[Som10] Ian Sommerville. *Software Engineering*. Addison-Wesley Publishing Company, USA, 9th edition, 2010.

[SRC10] Jeffrey Shafer, Scott Rixner, and Alan L Cox. The Hadoop distributed filesystem: Balancing portability and performance. In *2010 IEEE International Symposium on Performance Analysis of Systems & Software (ISPASS)*, pages 122–133. IEEE, March 2010.

[SS07] Kurt Schlegel and Bahvish Sood. Business Intelligence Platform Capability Matrix. Technical report, Gartner Group, 2007.

[SSV13] Fabian Schomm, Florian Stahl, and Gottfried Vossen. Marketplaces for data. *ACM SIGMOD Record*, 42(1):15, May 2013.

[Sta73] Herbert Stachowiak. *Allgemeine Modelltheorie*. Springer, Wien, New York, 1973.

[Sta03] Robert W Starinsky. *Maximizing business performance through software packages : best practices for justification, selection, and implementation*. Auerbach Publications, Boca Raton, FL, 2003.

[Sto14] Ion Stoica. Conquering big data with spark and BDAS. In *The 2014 ACM international conference on Measurement and modeling of computer systems - SIGMETRICS '14*, pages 193–193, New York, New York, USA, 2014. ACM Press.

[SVOK12] Frank Schönthaler, Gottfried Vossen, Andreas Oberweis, and Thomas Karle. *Business Processes for Business Communities*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[SZMR99] Rohini K. Srihari, Zhongfei Zhang, R. Manmatha, and Chandu Ravela. Multimedia indexing and retrieval. *ACM SIGIR Forum*, 33(1):34–35, 1999.

[SZS15] Zhaohao Sun, Huasheng Zou, and Kenneth Strang. Big Data Analytics as a Service for Business Intelligence. *IFIP International Federation for Information Processing*, 9373:200–211, 2015.

[Tan07] Andrew S Tanenbaum. *Modern Operating Systems*. Prentice Hall Press, Upper Saddle River, NJ, USA, 3rd edition, 2007.

[TDB+14] Ankit Toshniwal, Jake Donham, Nikunj Bhagat, Sailesh Mittal, Dmitriy Ryaboy, Siddarth Taneja, Amit Shukla, Karthik Ramasamy, Jignesh M. Patel, Sanjeev Kulkarni, Jason Jackson, Krishna Gade, and Maosong Fu. Storm@twitter. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data - SIGMOD '14*, pages 147–156, New York, New York, USA, 2014. ACM Press.

[Teu07] Rolf Alexander Teubner. Strategic Information Systems Planning: A Case Study from the Financial Services Industry. *The Journal of Strategic Information Systems*, 16(1):105–125, 2007.

[The95] The Standish Group. CHAOS Report. Technical report, 1995.

[The11] The Open Group. *TOGAF Version 9.1*. TOGAF Series. van Haren Publishing, 2011.

[TLH11] Maik Thiele, Wolfgang Lehner, and Dirk Habich. Data-Warehousing 3.0 – Die Rolle von Data-Warehouse- Systemen auf Basis von In-Memory-Technologie. In *Innovative Unternehmensanwendungen mit In-Memory Data Management (IMDM)*, pages 57–68, Mainz, 2011. Wolfgang Lehner, Gunther Piller.

[Tof71] Alvin Toffler. *Future Shock*. Bantam Books Non-Fiction. Bantam Books, 1971.

[TP13] Rolf Alexander Teubner and Alexander Robert Pellengahr. State of and perspectives for IS strategy research: A discussion paper. Working Papers, ERCIS - European Research Center for Information Systems 16, ERCIS, Münster, 2013.

[TS92] Geoffrey Towell and Jude W Shavlik. Interpretation of Artificial Neural Networks: Mapping Knowledge-Based Neural Networks into Rules. In J E Moody, S J Hanson, and R P Lippmann, editors, *Advances in Neural Information Processing Systems 4*, pages 977–984. Morgan-Kaufmann, 1992.

[TS16] Joao Tapadinhas and Bhavish Sood. Relaunch Mobile BI as Mobile Analytics to Achieve High-Impact Digital Business Outcomes. Technical report, Gartner Group, 2016.

[TSA+10] Ashish Thusoo, Zheng Shao, Suresh Anthony, Dhruba Borthakur, Namit Jain, Joydeep Sen Sarma, Raghotham Murthy, and Hao Liu. Data warehousing and analytics infrastructure at facebook. In *Proceedings of the 2010 international conference on Management of data - SIGMOD '10*, page 1013, New York, New York, USA, 2010. ACM Press.

[uRLA+16] Muhammad Habib ur Rehman, Chee Sun Liew, Assad Abbas, Prem Prakash Jayaraman, Teh Ying Wah, and Samee U. Khan. Big Data Reduction Methods: A Survey. *Data Science and Engineering*, 1(4):265–284, December 2016.

[vdA14] Wil M P van der Aalst. Data Scientist: The Engineer of the Future. In Kai Mertins, Frédérick Bénaben, Raúl Poler, and Jean-Paul Bourrières, editors, *Enterprise Interoperability VI*, pages 13–26, Cham, 2014. Springer International Publishing.

[VKI15] Peter C. Verhoef, P. K. Kannan, and J. Jeffrey Inman. From Multi-Channel Retailing to Omni-Channel Retailing. Introduction to the Special Issue on Multi-Channel Retailing. *Journal of Retailing*, 91(2):174–181, 2015.

[Vos91] Gottfried Vossen. *Data Models, Database Languages and Database Management Systems*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1991.

[Vos14] Gottfried Vossen. Big data as the new enabler in business and other intelligence. *Vietnam Journal of Computer Science*, 1(1):3–14, February 2014.

[vR14] Mark van Rijmenam. *Think Bigger: Developing a Successful Big Data Strategy for Your Business.* AMACOM, 2014.

[VSD17] Gottfried Vossen, Frank Schönthaler, and Stuart Dillon. *The Web at Graduation and Beyond.* Springer International Publishing, Cham, 2017.

[VSN+09] Jan Vom Brocke, Alexander Simons, Bjoern Niehaves, Bjorn Niehaves, and Kai Reimer. Reconstructing the giant: On the importance of rigour in documenting the literature search process. In *ECIS*, pages 2206–2217, Verona, Italy, 2009.

[WAM02] Michael Widenius, David Axmark, and MySQL AB. *MySQL Reference Manual.* O'Reilly Community Press, Sebastopol, CA, 1st edition, 2002.

[WB13] Karl Eugene Wiegers and Joy Beatty. *Software Requirements.* Microsoft Press, Redmond, WA, USA, 3rd edition, 2013.

[WCK+12] Hao Wang, Dogan Can, Abe Kazemzadeh, François Bar, and Shrikanth Narayanan. A System for Real-time Twitter Sentiment Analysis of 2012 U.S. Presidential Election Cycle. In *Proceedings of the ACL 2012 System Demonstrations*, ACL '12, pages 115–120, Stroudsburg, PA, USA, 2012. Association for Computational Linguistics.

[WFHP16] Ian H Witten, Eibe Frank, Mark A Hall, and Christopher J Pal. *Data Mining, Fourth Edition: Practical Machine Learning Tools and Techniques.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 4th edition, 2016.

[WH12] Thomas L. Wheelen and J. David Hunger. *Strategic management and business policy.* Pearson Prentice Hall, New Jersey, NY, USA, 13th edition, 2012.

[Whi15] Tom White. *Hadoop: The Definitive Guide.* O'Reilly, 4 edition, 2015.

[WKS+15] Guozhang Wang, Joel Koshy, Sriram Subramanian, Kartik Paramasivam, Mammad Zadeh, Neha Narkhede, Jun Rao, Jay Kreps, and Joe Stein. Building a Replicated Logging System with Apache Kafka. *Proceedings of the VLDB Endowment*, 8(12):1654–1655, 2015.

[WLRW15] Tim Weilkiens, Jesko G. Lamm, Stephan Roth, and Markus Walker. *Model-Based System Architecture*. John Wiley & Sons, Inc, Hoboken, NJ, USA, September 2015.

[WP02] John Ward and Joe Peppard. *Strategic Planning for Information Systems*. John Wiley & Sons, Inc., New York, NY, USA, 3rd edition, 2002.

[WRYK17] T Weise, M V Ramanath, D Yan, and K Knowles. *Learning Apache Apex: Real-time streaming applications with Apex*. Packt Publishing, Birmingham, UK, 2017.

[WSR+12] L Wu, R Sumbaly, C Riccomini, G Koo, H Kim, J Kreps, and S Shah. Avatara: OLAP for web-scale analytics products. In *Proceedings of the VLDB Endowment*, volume 5, pages 1874–1877, Istanbul, Turkey, 2012.

[YHF10] Philip S. Yu, Jiawei Han, and Christos Faloutsos, editors. *Link Mining: Models, Algorithms, and Applications*. Springer New York, New York, NY, 2010.

[YM15] Mobin Yasini and Guillaume Marchand. Toward a use case based classification of mobile health applications. *Studies in Health Technology and Informatics*, 210:175–179, 2015.

[Zac87] John A Zachman. A framework for information systems architecture. *IBM Systems Journal*, 26(3):276–292, September 1987.

[ZAL14] Reza Zafarani, Mohammad Ali Abbasi, and Huan Liu. *Social Media Mining: An Introduction*. Cambridge University Press, New York, NY, USA, 2014.

[Zar09] Dan Zarrella. *The Social Media Marketing Book*. O'Reilly Media, Inc., 1st edition, 2009.

[ZCD+12] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J Franklin, Scott Shenker, and Ion Stoica. Resilient Distributed Datasets: A Fault-tolerant Abstraction for In-memory Cluster Computing. In *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, NSDI'12, page 2, Berkeley, CA, USA, 2012. USENIX Association.

[ZDL+13] Matei Zaharia, Tathagata Das, Haoyuan Li, Timothy Hunter, Scott Shenker, and Ion Stoica. Discretized streams. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles - SOSP '13*, pages 423–438, New York, New York, USA, 2013. ACM Press.

[ZED+12] Paul Zikopoulos, Chris Eaton, Dirk DeRoos, Thomas Deutsch, and George Lapis. *Understanding Big Data: Analytics for Enterprise Class Hadoop and Streaming Data.* McGraw-Hill, New York, USA, 1st edition, 2012.

[ZR14] Ce Zhang and Christopher Ré. DimmWitted: a study of main-memory statistical analytics. *Proceedings of the VLDB Endowment*, 7(12):1283–1294, August 2014.

[ZTCO14] Hao Zhang, Bogdan Marius Tudor, Gang Chen, and Beng Chin Ooi. Efficient in-memory data management: an analysis. *Proceedings of the VLDB Endowment*, 7(10):833–836, June 2014.

List of Web Pages

[1] Adams, Tim: *Surge pricing comes to the supermarket | Technology | The Guardian.* https://www.theguardian.com/technology/2017/jun/04/surge-pricing-comes-to-the-supermarket-dynamic-personal-data. Version: 2017. – Last accessed: 2018-03-30

[2] Adrian, Merv: *IBM Ends Hadoop Distribution, Hortonworks Expands Hybrid Open Source - Merv Adrian.* https://blogs.gartner.com/merv-adrian/2017/06/21/ibm-ends-hadoop-distribution-hortonworks-expands-hybrid-open-source/. Version: 2017. – Last accessed: 2018-04-14

[3] Amatriain, Xavier ; Basilico, Justin: *System Architectures for Personalization and Recommendation.* https://medium.com/netflix-techblog/system-architectures-for-personalization-and-recommendation-e081aa94b5d8. Version: 2013. – Last accessed: 2018-04-12

[4] Anaconda Inc.: *Anaconda.* https://www.anaconda.com/. Version: 2018. – Last accessed: 2018-04-30

[5] Anaconda Inc.: *R language packages for Anaconda | Anaconda Documentation.* https://docs.anaconda.com/anaconda/packages/r-language-pkg-docs. Version: 2018. – Last accessed: 2018-04-30

[6] Andlinger, Paul: *RDBMS dominate the database market, but NoSQL systems are catching up.* https://db-engines.com/en/blog_post/23. Version: 2013. – Last accessed: 2018-03-05

[7] Andlinger, Paul: *Multi-model database systems – a new trend!* https://db-engines.com/de/blog_post/29. Version: 2014. – Last accessed: 2018-04-13

[8] Apache NiFi Team: *Apache NiFi Documentation.* https://nifi.apache.org/docs.html. Version: 2018. – Last accessed: 2018-05-01

[9] Apache Software Foundation: *Apache Flume - Performance Tuning - part 1.* https://blogs.apache.org/flume/. Version: 2012. – Last accessed: 2018-04-11

[10] Apache Software Foundation: *Apache SAMOA*. https://samoa.incubator.apache.org/. Version: 2014. – Last accessed: 2018-04-11

[11] Apache Software Foundation: *Apache SAMOA Documentation*. https://samoa.incubator.apache.org/documentation/Home.html. Version: 2014. – Last accessed: 2018-04-30

[12] Apache Software Foundation: *S4 Incubation Status - Apache Incubator*. http://incubator.apache.org/projects/s4.html. Version: 2014. – Last accessed: 2018-04-11

[13] Apache Software Foundation: *Apache Storm*. http://storm.apache.org/. Version: 2015. – Last accessed: 2018-04-10

[14] Apache Software Foundation: *Chapter 14. Apache HBase (TM) Operational Management*. http://hbase.apache.org/0.94/book/ops_mgt.html. Version: 2015. – Last accessed: 2018-05-01

[15] Apache Software Foundation: *Apache Kafka 0.10.0.0 Changelog*. https://archive.apache.org/dist/kafka/0.10.0.0/RELEASE_NOTES.html. Version: 2016. – Last accessed: 2018-04-10

[16] Apache Software Foundation: *Apache Mahout - Release Notes*. https://mahout.apache.org/general/release-notes.html. Version: 2016. – Last accessed: 2018-04-15

[17] Apache Software Foundation: *Apache Apex Malhar - apex/malhar/contrib/ - GitHub*. https://github.com/apache/apex-malhar/tree/master/contrib/src/main/java/org/apache/apex/malhar/contrib. Version: 2017. – Last accessed: 2018-04-11

[18] Apache Software Foundation: *Apache Flink: Ecosystem*. https://flink.apache.org/ecosystem.html. Version: 2017. – Last accessed: 2018-04-10

[19] Apache Software Foundation: *Apache Flink: Introduction to Apache Flink®*. https://flink.apache.org/introduction.html. Version: 2017. – Last accessed: 2018-04-10

[20] Apache Software Foundation: *Apache Flink: Scalable Stream and Batch Data Processing*. https://flink.apache.org/. Version: 2017. – Last accessed: 2018-04-10

[21] Apache Software Foundation: *Apache Gearpump (Incubating): Overview*. https://gearpump.apache.org/overview.html. Version: 2017. – Last accessed: 2018-04-11

[22] Apache Software Foundation: *Apache Hadoop 2.9.0 - HDFS High Availability*. https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs/HDFSHighAvailabilityWithNFS.html. Version: 2017. – Last accessed: 2018-03-26

[23] Apache Software Foundation: *Apache Kafka*. https://kafka.apache.org/. Version: 2017. – Last accessed: 2018-04-10

[24] Apache Software Foundation: *Apache Kafka - Documentation # Design*. https://kafka.apache.org/documentation/#design. Version: 2017. – Last accessed: 2018-04-10

[25] Apache Software Foundation: *Apache Kafka - Introduction*. https://kafka.apache.org/intro. Version: 2017. – Last accessed: 2018-04-10

[26] Apache Software Foundation: *Apache Kafka - Kafka Streams - Core Concepts*. https://kafka.apache.org/11/documentation/streams/core-concepts. Version: 2017. – Last accessed: 2018-04-10

[27] Apache Software Foundation: *Apache Kafka - Kafka Streams - Introduction*. https://kafka.apache.org/11/documentation/streams/. Version: 2017. – Last accessed: 2018-04-10

[28] Apache Software Foundation: *Apache Mahout - Algorithms*. https://mahout.apache.org/users/basics/algorithms.html. Version: 2017. – Last accessed: 2018-04-30

[29] Apache Software Foundation: *Apache NiFi*. https://nifi.apache.org/. Version: 2017. – Last accessed: 2018-04-11

[30] Apache Software Foundation: *CQL*. https://cassandra.apache.org/doc/old/CQL-3.0.html. Version: 2017. – Last accessed: 2018-04-14

[31] Apache Software Foundation: *Flume 1.8.0 User Guide — Apache Flume*. https://flume.apache.org/FlumeUserGuide.html. Version: 2017. – Last accessed: 2018-04-11

[32] Apache Software Foundation: *Intro to Cooccurrence Recommenders with Spark*. http://mahout.apache.org/users/recommender/intro-cooccurrence-spark.html. Version: 2017. – Last accessed: 2018-04-15

[33] Apache Software Foundation: *MADlib*. http://madlib.apache.org/. Version: 2017. – Last accessed: 2018-04-30

[34] Apache Software Foundation: *MADlib: Main Page | Documentation.* http://madlib.apache.org/docs/latest/index.html. Version: 2017. – Last accessed: 2018-04-30

[35] Apache Software Foundation: *Open source memory-centric distributed database, caching, and processing platform - Apache Ignite™.* https://ignite.apache.org/. Version: 2017. – Last accessed: 2018-04-11

[36] Apache Software Foundation: *Parallel Frequent Pattern Mining.* https://mahout.apache.org/users/misc/parallel-frequent-pattern-mining.html. Version: 2017. – Last accessed: 2018-04-30

[37] Apache Software Foundation: *Powered By - Hadoop Wiki.* https://wiki.apache.org/hadoop/PoweredBy. Version: 2017. – Last accessed: 2018-03-26

[38] Apache Software Foundation: *Supported Features: Apache Hive 2.3 - Apache Hive - Apache Software Foundation.* https://cwiki.apache.org/confluence/display/Hive/Supported+Features%3A+Apache+Hive+2.3. Version: 2017. – Last accessed: 2018-04-15

[39] Apache Software Foundation: *Supported Features: Apache Hive 2.3 - Apache Hive - Apache Software Foundation.* https://cwiki.apache.org/confluence/display/Hive/Supported+Features%3A+Apache+Hive+2.3. Version: 2017. – Last accessed: 2018-03-28

[40] Apache Software Foundation: *Welcome to Apache Flume — Apache Flume.* https://flume.apache.org/. Version: 2017. – Last accessed: 2018-04-10

[41] Apache Software Foundation: *2.2. Connectors — Apache Sqoop documentation.* http://sqoop.apache.org/docs/1.99.7/user/Connectors.html. Version: 2018. – Last accessed: 2018-04-15

[42] Apache Software Foundation: *Apache Apex.* https://apex.apache.org/. Version: 2018. – Last accessed: 2018-04-11

[43] Apache Software Foundation: *Apache Apex Malhar - apex/malhar/lib - GitHub.* https://github.com/apache/apex-malhar/tree/master/library/src/main/java/org/apache/apex/malhar/lib. Version: 2018. – Last accessed: 2018-04-11

[44] Apache Software Foundation: *Apache Apex Malhar Documentation.* https://apex.apache.org/docs/malhar/. Version: 2018. – Last accessed: 2018-04-11

[45] Apache Software Foundation: *Apache Beam*. https://beam.apache.org/. Version: 2018. – Last accessed: 2018-04-11

[46] Apache Software Foundation: *Apache Calcite - Streaming*. https://calcite.apache.org/docs/stream.html. Version: 2018. – Last accessed: 2018-04-10

[47] Apache Software Foundation: *Apache Cassandra*. http://cassandra.apache.org/. Version: 2018. – Last accessed: 2018-04-30

[48] Apache Software Foundation: *Apache Cassandra - Documentation - Aggregate functions*. http://cassandra.apache.org/doc/latest/cql/functions.html#aggregate-functions. Version: 2018. – Last accessed: 2018-04-30

[49] Apache Software Foundation: *Apache Cassandra - Documentation - CQL*. http://cassandra.apache.org/doc/latest/cql/index.html. Version: 2018. – Last accessed: 2018-04-30

[50] Apache Software Foundation: *Apache Flink 1.4 Documentation: Command-Line Interface*. https://ci.apache.org/projects/flink/flink-docs-release-1.4/ops/cli.html. Version: 2018. – Last accessed: 2018-04-30

[51] Apache Software Foundation: *Apache Flink 1.4 Documentation: Connectors*. https://ci.apache.org/projects/flink/flink-docs-release-1.4/dev/batch/connectors.html. Version: 2018. – Last accessed: 2018-04-30

[52] Apache Software Foundation: *Apache Flink 1.4 Documentation: Data Streaming Fault Tolerance*. https://ci.apache.org/projects/flink/flink-docs-release-1.4/internals/stream_checkpointing.html. Version: 2018. – Last accessed: 2018-04-10

[53] Apache Software Foundation: *Apache Flink 1.4 Documentation: DataSet Transformations*. https://ci.apache.org/projects/flink/flink-docs-release-1.4/dev/batch/dataset_transformations.html. Version: 2018. – Last accessed: 2018-04-30

[54] Apache Software Foundation: *Apache Flink 1.4 Documentation: FlinkML - Machine Learning for Flink*. https://ci.apache.org/projects/flink/flink-docs-release-1.4/dev/libs/ml/. Version: 2018. – Last accessed: 2018-04-30

[55] Apache Software Foundation: *Apache Flink 1.4 Documentation: FlinkML - Machine Learning for Flink*. https://ci.apache.org/projects/flink/flink-docs-release-1.4/dev/libs/ml/. Version: 2018. – Last accessed: 2018-04-10

[56] Apache Software Foundation: *Apache Flink 1.4 Documentation: Operators*. https://ci.apache.org/projects/flink/flink-docs-release-1.4/dev/stream/operators/. Version: 2018. – Last accessed: 2018-04-30

[57] Apache Software Foundation: *Apache Flink 1.4 Documentation: Streaming Connectors*. https://ci.apache.org/projects/flink/flink-docs-release-1.4/dev/connectors/. Version: 2018. – Last accessed: 2018-04-30

[58] Apache Software Foundation: *Apache Flink 1.4 Documentation: Table API & SQL*. https://ci.apache.org/projects/flink/flink-docs-release-1.4/dev/table/index.html. Version: 2018. – Last accessed: 2018-04-10

[59] Apache Software Foundation: *Apache Flink 1.4 Documentation: Table Sources and Sinks*. https://ci.apache.org/projects/flink/flink-docs-release-1.4/dev/table/sourceSinks.html#jdbcappendtablesink. Version: 2018. – Last accessed: 2018-05-01

[60] Apache Software Foundation: *Apache Hadoop 3.0.1 - Apache Hadoop YARN*. https://hadoop.apache.org/docs/r3.0.1/hadoop-yarn/hadoop-yarn-site/YARN.html. Version: 2018. – Last accessed: 2018-03-27

[61] Apache Software Foundation: *Apache Hadoop 3.0.1 - C API libhdfs*. http://hadoop.apache.org/docs/r3.0.1/hadoop-project-dist/hadoop-hdfs/LibHdfs.html. Version: 2018. – Last accessed: 2018-03-27

[62] Apache Software Foundation: *Apache Hadoop 3.0.1 - File System Shell - Overview*. https://hadoop.apache.org/docs/r3.0.1/hadoop-project-dist/hadoop-common/FileSystemShell.html. Version: 2018. – Last accessed: 2018-03-27

[63] Apache Software Foundation: *Apache Hadoop 3.0.1 - HDFS Architecture Guide*. http://hadoop.apache.org/docs/r3.0.1/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html. Version: 2018. – Last accessed: 2018-03-24

[64] Apache Software Foundation: *Apache Hadoop 3.0.1 - HDFS Commands Guide*. http://hadoop.apache.org/docs/r3.0.1/hadoop-project-dist/hadoop-hdfs/HDFSCommands.html. Version: 2018. – Last accessed: 2018-03-27

[65] Apache Software Foundation: *Apache Hadoop 3.0.1 - HDFS Federation*. https://hadoop.apache.org/docs/r3.0.1/hadoop-project-dist/hadoop-hdfs/Federation.html. Version: 2018. – Last accessed: 2018-03-27

[66] Apache Software Foundation: *Apache Hadoop 3.0.1 - MapReduce Tutorial*. https://hadoop.apache.org/docs/r3.0.1/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html. Version: 2018. – Last accessed: 2018-03-28

[67] Apache Software Foundation: *Apache Hadoop 3.0.1 - ViewFs Guide*. https://hadoop.apache.org/docs/r3.0.1/hadoop-project-dist/hadoop-hdfs/ViewFs.html. Version: 2018. – Last accessed: 2018-03-27

[68] Apache Software Foundation: *Apache Hadoop Releases*. http://hadoop.apache.org/releases.html. Version: 2018. – Last accessed: 2018-03-27

[69] Apache Software Foundation: *Apache HBase – Apache HBase™ Home*. https://hbase.apache.org/. Version: 2018. – Last accessed: 2018-04-30

[70] Apache Software Foundation: *Apache HBase – Apache HBase (TM) ACID Properties*. https://hbase.apache.org/acid-semantics.html. Version: 2018. – Last accessed: 2018-04-30

[71] Apache Software Foundation: *Apache HBase ™ Reference Guide*. https://hbase.apache.org/book.html. Version: 2018. – Last accessed: 2018-04-30

[72] Apache Software Foundation: *Apache Kafka*. https://kafka.apache.org/11/documentation/streams/developer-guide/dsl-api.html. Version: 2018. – Last accessed: 2018-04-30

[73] Apache Software Foundation: *Apache Kafka - Running Streams Applications*. http://kafka.apache.org/11/documentation/streams/developer-guide/running-app.html. Version: 2018. – Last accessed: 2018-04-09

[74] Apache Software Foundation: *Apache Kafka - Streams - Developer Guide*. https://kafka.apache.org/0102/documentation/streams/developer-guide. Version: 2018. – Last accessed: 2018-04-09

[75] Apache Software Foundation: *Apache Mahout*. https://mahout.apache.org/. Version: 2018. – Last accessed: 2018-04-30

[76] Apache Software Foundation: *Apache Spark™ - Lightning-Fast Cluster Computing*. https://spark.apache.org/. Version: 2018. – Last accessed: 2018-04-10

[77] Apache Software Foundation: *Apache Storm - Concepts*. http://storm.apache.org/releases/1.2.1/Concepts.html. Version: 2018. – Last accessed: 2018-04-30

[78] Apache Software Foundation: *Apache Storm - Documentation*. http://storm.apache.org/releases/1.2.1/index.html. Version: 2018. – Last accessed: 2018-04-30

[79] Apache Software Foundation: *Applications - Apache Apex Documentation*. http://apex.apache.org/docs/apex/application_development/. Version: 2018. – Last accessed: 2018-04-11

[80] Apache Software Foundation: *Beam Capability Matrix*. https://beam.apache.org/documentation/runners/capability-matrix/. Version: 2018. – Last accessed: 2018-03-28

[81] Apache Software Foundation: *Cluster Mode Overview - Spark 2.3.0 Documentation*. https://spark.apache.org/docs/latest/cluster-overview.html. Version: 2018. – Last accessed: 2018-04-10

[82] Apache Software Foundation: *Configuration - Apache Gearpump(incubating)*. https://gearpump.apache.org/releases/latest/deployment/deployment-configuration/index.html. Version: 2018. – Last accessed: 2018-04-11

[83] Apache Software Foundation: *CSV Import Guide: How to Use LOAD CSV Command for Neo4j*. https://neo4j.com/developer/guide-import-csv/. Version: 2018. – Last accessed: 2018-05-01

[84] Apache Software Foundation: *Design Your Pipeline*. https://beam.apache.org/documentation/pipelines/design-your-pipeline/. Version: 2018. – Last accessed: 2018-03-28

[85] Apache Software Foundation: *Distributed Linear Algebra*. http://mahout.apache.org/docs/latest/algorithms/linear-algebra/. Version: 2018. – Last accessed: 2018-04-30

[86] Apache Software Foundation: *Ecosystem - Apache Kafka - Apache Software Foundation*. https://cwiki.apache.org/confluence/display/KAFKA/Ecosystem. Version: 2018. – Last accessed: 2018-04-30

[87] Apache Software Foundation: *Home - Apache Hive - Apache Software Foundation*. https://cwiki.apache.org/confluence/display/Hive/Home. Version: 2018. – Last accessed: 2018-04-15

[88] Apache Software Foundation: *Home - Apache Hive - Apache Software Foundation*. https://cwiki.apache.org/confluence/display/Hive/Home. Version: 2018. – Last accessed: 2018-03-28

[89] Apache Software Foundation: *IBasicBolt (Storm 1.2.0 API)*. http://storm.apache.org/releases/1.2.1/javadocs/org/apache/storm/topology/IBasicBolt.html. Version: 2018. – Last accessed: 2018-04-30

[90] Apache Software Foundation: *IRichBolt (Storm 1.2.0 API)*. http://storm.apache.org/releases/1.2.1/javadocs/org/apache/storm/topology/IRichBolt.html. Version: 2018. – Last accessed: 2018-04-30

[91] Apache Software Foundation: *MLlib | Apache Spark*. https://spark.apache.org/mllib/. Version: 2018. – Last accessed: 2018-04-10

[92] Apache Software Foundation: *MLlib: Main Guide - Spark 2.3.0 Documentation*. https://spark.apache.org/docs/latest/ml-guide.html. Version: 2018. – Last accessed: 2018-04-30

[93] Apache Software Foundation: *Preprocesors*. http://mahout.apache.org/docs/latest/algorithms/preprocessors/. Version: 2018. – Last accessed: 2018-04-30

[94] Apache Software Foundation: *Releases - apache/hbase - GitHub*. https://github.com/apache/hbase/releases. Version: 2018. – Last accessed: 2018-04-30

[95] Apache Software Foundation: *Samza*. http://samza.apache.org/. Version: 2018. – Last accessed: 2018-04-11

[96] Apache Software Foundation: *Samza - Windowing*. https://samza.apache.org/learn/documentation/0.7.0/container/windowing.html. Version: 2018. – Last accessed: 2018-05-02

[97] Apache Software Foundation: *Spark Streaming - Spark 2.3.0 Documentation*. https://spark.apache.org/docs/latest/streaming-programming-guide.html. Version: 2018. – Last accessed: 2018-04-10

[98] Apache Software Foundation: *Spark Streaming | Apache Spark*. https://spark.apache.org/streaming/. Version: 2018. – Last accessed: 2018-04-10

[99] Apache Software Foundation: *Sqoop - Home*. http://sqoop.apache.org/. Version: 2018. – Last accessed: 2018-04-30

[100] Apache Software Foundation: *Sqoop User Guide (v1.4.7)*. http://sqoop.apache.org/docs/1.4.7/SqoopUserGuide.html. Version: 2018. – Last accessed: 2018-04-30

[101] Apache Software Foundation: *Structured Streaming + Kafka Integration Guide (Kafka broker version 0.10.0 or higher) - Spark 2.2.0 Documentation*. https://spark.apache.org/docs/2.2.0/structured-streaming-kafka-integration.html. Version: 2018. – Last accessed: 2018-05-01

[102] Apache Software Foundation: *Trident Tutorial*. http://storm.apache.org/releases/2.0.0-SNAPSHOT/Trident-tutorial.html. Version: 2018. – Last accessed: 2018-04-10

[103] Apache Software Foundation: *Welcome to Apache™ Hadoop®!* http://hadoop.apache.org/. Version: 2018. – Last accessed: 2018-04-14

[104] AppBrain: *Number of available Android applications - AppBrain*. http://www.appbrain.com/stats/number-of-android-apps. Version: 2018. – Last accessed: 2018-04-17

[105] Asay, Matt: *Why the world's largest Hadoop installation may soon become the norm - TechRepublic*. https://www.techrepublic.com/article/why-the-worlds-largest-hadoop-installation-may-soon-become-the-norm/. Version: 2014. – Last accessed: 2018-03-26

[106] Balasubramanyan, Arun: *Data warehouse augmentation, Part 3: Use big data technology for an active archive*. https://www.ibm.com/developerworks/analytics/library/ba-augment-data-warehouse3/index.html?ca=drs-. Version: 2014. – Last accessed: 2018-04-15

[107] Bani, Moti: *Page File – The definitive guide – Windows Security*. https://blogs.technet.microsoft.com/motiba/2015/10/15/page-file-the-definitive-guide/. Version: 2015. – Last accessed: 2018-04-24

[108] Barr, Jeff: *Now Available – EC2 Instances with 4 TB of Memory | AWS News Blog*. https://aws.amazon.com/de/blogs/aws/now-available-ec2-instances-with-4-tb-of-memory/. Version: 2017. – Last accessed: 2018-04-16

[109] Barrasa, Jesús: *RDF Triple Stores vs. Labeled Property Graphs: What's the Difference?* https://neo4j.com/blog/rdf-triple-store-vs-labeled-property-graph-difference/. Version: 2017. – Last accessed: 2018-04-14

[110] Basho Technologies: *Riak KV - Advanced MapReduce*. http://docs.basho.com/riak/kv/2.2.3/developing/app-guide/advanced-mapreduce/. Version: 2017. – Last accessed: 2018-04-30

[111] Basho Technologies: *Riak KV - Basho*. http://basho.com/products/riak-kv/. Version: 2017. – Last accessed: 2018-04-01

[112] Basho Technologies: *Riak KV - Using Secondary Indexes (2i).* http://docs.basho.com/riak/kv/2.2.3/developing/usage/secondary-indexes/. Version: 2017. – Last accessed: 2018-04-30

[113] Basho Technologies: *Riak KV Key/Value Modeling.* http://docs.basho.com/riak/kv/2.2.3/developing/key-value-modeling/. Version: 2017. – Last accessed: 2018-04-30

[114] Basho Technologies: *Using Search.* https://docs.basho.com/riak/kv/2.2.3/developing/usage/search/. Version: 2017. – Last accessed: 2018-04-30

[115] Batchelor, Mark: *Latest Pentaho Data Integration (aka Kettle) Documentation - Pentaho Data Integration - Pentaho Wiki.* https://wiki.pentaho.com/display/EAI/Latest+Pentaho+Data+Integration+(aka+Kettle)+Documentation. Version: 2015. – Last accessed: 2018-04-30

[116] Beck, Kent ; Beedle, Mike ; Bennekum, Arie van ; Cockburn, Alistair ; Cunningham, Ward ; Fowler, Martin ; Grenning, James ; Highsmith, Jim ; Hunt, Anrew ; Jeffries, Ron ; Kern, Jon ; Marick, Brian ; Martin, Robert C. ; Mellor, Steve ; Schwaber, Ken ; Sutherland, Jeff ; Thomas, David: *Manifesto for Agile Software Development.* http://agilemanifesto.org/. Version: 2001. – Last accessed: 2018-05-06

[117] Benningfield, Damond: *Ground Proximity Warnings | How Things Work | Air & Space Magazine.* https://www.airspacemag.com/how-things-work/ground-proximity-warnings-4244883/. Version: 2003. – Last accessed: 2018-04-08

[118] Bersin, Josh: *Oracle Transformed - It's Now All about the Cloud.* https://www.forbes.com/sites/joshbersin/2012/10/03/oracle-openworld-its-now-all-about-the-cloud/#31edd7a55ab3. Version: 2012. – Last accessed: 2018-03-29

[119] Biyikoglu, Cihan: *CAP Theorem and Couchbase Server... But this time with XDCR | The Couchbase Blog.* https://blog.couchbase.com/cap-theorem-and-couchbase-server-time-xdcr/. Version: 2014. – Last accessed: 2018-05-02

[120] Bloomberg, Jason: *Oracle's Cloud Strategy: Ruthless Or 'Byzantine'?* https://www.forbes.com/sites/jasonbloomberg/2017/07/11/oracles-cloud-strategy-ruthless-or-byzantine/#1ea55cd762d9. Version: 2017. – Last accessed: 2018-03-29

[121] Bonaci, Marko: *The history of Hadoop - Marko Bonaci - Medium.* https://medium.com/@markobonaci/the-history-of-hadoop-68984a11704. Version: 2015. – Last accessed: 2018-03-27

[122] Borne, Kirk: *To 10 Big Data Challenges - A Serious Look at 10 Big Data V's | MapR*. https://mapr.com/blog/top-10-big-data-challenges-serious-look-10-big-data-vs/. Version: 2014. – Last accessed: 2018-03-19

[123] Bradshaw, Tim: *Snapchat to stop retaining location data on under-16s in Europe*. https://www.ft.com/content/f8a21928-48f0-11e8-8ee8-cae73aab7ccb. Version: 2018. – Last accessed: 2018-05-07

[124] Brownlee, Jason: *What is the Difference Between Test and Validation Datasets?* https://machinelearningmastery.com/difference-test-validation-datasets/. Version: 2017. – Last accessed: 2018-05-03

[125] Butler, Brandon: *The top 5 Hadoop distributions, according to Forrester | Network World*. https://www.networkworld.com/article/3024812/big-data-business-intelligence/the-top-5-hadoop-distributions-according-to-forrester.html. Version: 2016. – Last accessed: 2018-04-14

[126] Cafarella, Michael: *NutchDistributedFileSystem - Nutch Wiki*. https://wiki.apache.org/nutch/NutchDistributedFileSystem. Version: 2009. – Last accessed: 2018-03-27

[127] CallidusCloud / SAP: *Basic Concepts · OrientDB Manual*. https://orientdb.com/docs/last/Concepts.html. Version: 2018. – Last accessed: 2018-04-13

[128] Cannon, Paul: *Simple data importing and exporting with Cassandra | DataStax*. https://www.datastax.com/dev/blog/simple-data-importing-and-exporting-with-cassandra. Version: 2012. – Last accessed: 2018-05-01

[129] Casters, Matt ; Wilkerson, Philipp: *Pentaho Data Integration Steps - Pentaho Data Integration - Pentaho Wiki*. https://wiki.pentaho.com/display/EAI/Pentaho+Data+Integration+Steps. Version: 2018. – Last accessed: 2018-04-30

[130] Central Intelligence Agency: *About CIA - Central Intelligence Agency*. https://www.cia.gov/about-cia. Version: 2013. – Last accessed: 2018-03-05

[131] Chan, Samantha: *Streams Quick Start Guide - Streamsdev*. https://developer.ibm.com/streamsdev/docs/streams-quick-start-guide/. Version: 2015. – Last accessed: 2017-08-23

[132] Chawda, Bhupesh: *Machine Learning on Apache Apex with Apache SAMOA - DataTorrent*. https://www.datatorrent.com/blog/machine-learning-apache-apex-apache-samoa/. Version: 2016. – Last accessed: 2018-04-11

[133] Chowdury, Sandip: *Data warehouse augmentation, Part 1: Big data and data warehouse augmentation.* https://www.ibm.com/developerworks/library/ba-augment-data-warehouse1/index.html. Version: 2014. – Last accessed: 2018-04-15

[134] Clarke, Matt: *Are there any limits to horizontally scaling Nifi? - Hortonworks.* https://community.hortonworks.com/questions/142058/are-there-any-limits-to-horizontally-scaling-nifi.html. Version: 2017. – Last accessed: 2018-04-30

[135] Cloudera: *Apache Hadoop core components | Cloudera.* https://www.cloudera.com/products/open-source/apache-hadoop/hdfs-mapreduce-yarn.html. Version: 2018. – Last accessed: 2018-04-15

[136] Cloudera: *Apache Hadoop open source ecosystem | Cloudera.* https://www.cloudera.com/products/open-source/apache-hadoop.html. Version: 2018. – Last accessed: 2018-04-14

[137] Cloudera: *Supported Sources, Sinks, and Channels | 5.5.x | Cloudera Documentation.* https://www.cloudera.com/documentation/enterprise/5-5-x/topics/cdh_ig_flume_supported_sources_sinks_channels.html. Version: 2018. – Last accessed: 2018-04-11

[138] CNET: *Twitter's huge bot problem is out of the bag | CNET.* https://www.cnet.com/news/twitter-bot-fake-followers-problem-out-of-the-bag/. Version: 2018. – Last accessed: 2018-03-19

[139] Cockroach Labs: *Build a Java App with CockroachDB | CockroachDB.* https://www.cockroachlabs.com/docs/stable/build-a-java-app-with-cockroachdb.html. Version: 2018. – Last accessed: 2018-05-01

[140] Cockroach Labs: *Cockroach Labs - Cockroach DB.* https://www.cockroachlabs.com/. Version: 2018. – Last accessed: 2018-04-30

[141] Cockroach Labs: *Detailed SQL Standard Comparison | CockroachDB.* https://www.cockroachlabs.com/docs/stable/detailed-sql-support.html. Version: 2018. – Last accessed: 2018-04-30

[142] Cockroach Labs: *Frequently Asked Questions | CockroachDB.* https://www.cockroachlabs.com/docs/stable/frequently-asked-questions.html. Version: 2018. – Last accessed: 2018-04-25

[143] Cockroach Labs: *SQL Feature Support in CockroachDB v2.0 | CockroachDB.* https://www.cockroachlabs.com/docs/stable/sql-feature-support.html. Version: 2018. – Last accessed: 2018-04-30

[144] Confluent, Inc.: *Kafka Connect | Confluent.* https://www.confluent.io/product/connectors/. Version: 2018. – Last accessed: 2018-04-30

[145] Confluent Inc.: *Kafka FileStream Connectors — Confluent Platform.* https://docs.confluent.io/current/connect/connect-filestream/filestream_connector.html#. Version: 2018. – Last accessed: 2018-05-01

[146] Costello, Sam: *How Many Apps Are in Apple's App Store (March 2018).* https://www.lifewire.com/how-many-apps-in-app-store-2000252. Version: 2018. – Last accessed: 2018-04-17

[147] Couchbase: *query/n1ql-select.md at master - couchbase/query - GitHub.* https://github.com/couchbase/query/blob/master/docs/n1ql-select.md. Version: 2017. – Last accessed: 2018-04-30

[148] Couchbase: *Coucbase - Aggregate Functions.* https://developer.couchbase.com / documentation / server / current / n1ql / n1ql - language - reference / aggregatefun.html. Version: 2018. – Last accessed: 2018-04-30

[149] Couchbase: *Couchbase - Data Access.* https://developer.couchbase.com/documentation/server/current/data-access/data-access-intro.html. Version: 2018. – Last accessed: 2018-04-30

[150] Couchbase: *Couchbase - Querying with N1QL # Indexes.* https://developer.couchbase.com/documentation/server/current/sdk/n1ql-query.html#story-h2-8. Version: 2018. – Last accessed: 2018-04-30

[151] Couchbase: *Couchbase - Release Notes.* https://developer.couchbase.com/documentation/server/current/release-notes/relnotes.html. Version: 2018. – Last accessed: 2018-04-30

[152] Couchbase: *NoSQL Engagement Database | Couchbase.* https://www.couchbase.com/. Version: 2018. – Last accessed: 2018-04-30

[153] Damji, Jules: *DataFrames, RDDs, and Datasets: A Tale of Three Apache Spark APIs.* https://databricks.com/blog/2016/07/14/a-tale-of-three-apache-spark-apis-rdds-dataframes-and-datasets.html. Version: 2016. – Last accessed: 2018-04-20

[154] Data Mining Group (DMG): *PMML 4.3 - General Structure.* http://dmg.org/pmml/v4-3/GeneralStructure.html. Version: 2016. – Last accessed: 2018-04-20

[155] Databricks: *Spark Packages.* https://spark-packages.org/?q=tags%3A%22DataSources%22. Version: 2018. – Last accessed: 2018-05-01

[156] DataStax: *CQL limits | CQL for Cassandra 3.0.* https://docs.datastax.com/en/cql/3.3/cql/cql_reference/refLimits.html. Version: 2018. – Last accessed: 2018-04-30

[157] DataStax Inc.: *How is the serial consistency level configured? | Apache Cassandra 3.0.* https://docs.datastax.com/en/cassandra/3.0/cassandra/dml/dmlConfigSerialConsistency.html. Version: 2018. – Last accessed: 2018-04-14

[158] Dell Corporation: *Dell EMC PowerEdge R940 Rack Server | Dell United States.* http://www.dell.com/en-us/work/shop/povw/poweredge-r940. Version: 2018. – Last accessed: 2018-04-24

[159] Der Beauftragte der Bundesregierung für Informationstechnik: *IT-Beauftragter der Bundesregierung | V-Modell XT.* https://www.cio.bund.de/Web/DE/Architekturen-und-Standards/V-Modell-XT/vmodell_xt_node.html. Version: 2005. – Last accessed: 2018-05-05

[160] Desjardins, Jeff: *Infographic: What Happens in an Internet Minute in 2017?* http://www.visualcapitalist.com/happens-internet-minute-2017/. Version: 2017. – Last accessed: 2018-03-20

[161] developers, Scikit learn: *2.7. Novelty and Outlier Detection — scikit-learn 0.19.1 documentation.* http://scikit-learn.org/stable/modules/outlier_detection.html. Version: 2018. – Last accessed: 2018-04-30

[162] Dresner Advisory Services LLC: *Dresner Advisory Services - In the news - Top Resources for Business Intelligence Solution Selection.* http://dresneradvisory.com/in-the-news/13. Version: 2006. – Last accessed: 2018-02-14

[163] Dumbill, Edd: *Understanding the Data Value Chain.* http://www.ibmbigdatahub.com/blog/understanding-data-value-chain. Version: 2014. – Last accessed: 2017-09-22

[164] Dutcher, Jennifer: *What is Big Data - Blog.* https://datascience.berkeley.edu/what-is-big-data. Version: 2014. – Last accessed: 2018-03-18

[165] Edlich, Stefan: *NOSQL Databases.* http://www.nosql-database.org/. Version: 2018. – Last accessed: 2018-04-12

[166] Einstein, Dave: *Big Data Needs Big Storage: Where to Keep Gigabytes, Terabytes and Petabytes of Data.* https://www.forbes.com/sites/netapp/2012/08/15/big-data-needs-big-storage-where-to-keep-gigabytes-terabytes-and-petabytes-of-data. Version: 2012. – Last accessed: 2018-03-16

[167] EMarketer: *Slowing Growth Ahead for Worldwide Internet Audience - eMarketer*. https://www.emarketer.com/Article/Slowing-Growth-Ahead-Worldwide-Internet-Audience/1014045. Version: 2016. – Last accessed: 2018-04-17

[168] Erwen, Stephan: *FLIP-6 - Flink Deployment and Process Model - Standalone, Yarn, Mesos, Kubernetes, etc. - Apache Flink - Apache Software Foundation*. https://cwiki.apache.org/confluence/pages/viewpage.action?pageId=65147077. Version: 2018. – Last accessed: 2018-04-19

[169] European Commission: *Data protection | European Commission*. https://ec.europa.eu/info/law/law-topic/data-protection_en. Version: 2018. – Last accessed: 2018-05-07

[170] Foo, Andrew: *Is the Data Warehouse Dead? | IBM Big Data & Analytics Hub*. http://www.ibmbigdatahub.com/blog/data-warehouse-dead. Version: 2013. – Last accessed: 2018-05-08

[171] Foundation, Apache S.: *Apache Cassandra - Documentation - Data Manipulation*. http://cassandra.apache.org/doc/latest/cql/dml.html?highlight=join. Version: 2018. – Last accessed: 2018-04-30

[172] Fox, Dominic: *How Not To Use Cassandra Like An RDBMS (and what will happen if you do) - OpenCredo*. https://opencredo.com/how-not-to-use-cassandra-like-an-rdbms-and-what-will-happen-if-you-do/. Version: 2016. – Last accessed: 2018-04-14

[173] Gartner Group: *Relaunch Mobile BI as Mobile Analytics to Achieve High-Impact Digital Business Outcomes*. https://www.gartner.com/doc/3394517/relaunch-mobile-bi-mobile-analytics. Version: 2016. – Last accessed: 2018-04-17

[174] Gartner Group: *Advanced Analytics - Big Data Analytics Defined by Gartner*. https://www.gartner.com/it-glossary/advanced-analytics/. Version: 2017. – Last accessed: 2018-03-30

[175] Gartner Group: *Big Data Analytics - Predictive Analytics - Gartner Glossary*. https://www.gartner.com/it-glossary/predictive-analytics/. Version: 2017. – Last accessed: 2018-03-29

[176] Gartner Group: *Gartner Says Worldwide Business Intelligence and Analytics Market to Reach $18.3 Billion in 2017*. https://www.gartner.com/newsroom/id/3612617. Version: 2017. – Last accessed: 2018-03-30

[177] Gartner Group: *Prescriptive Analytics - Gartner IT Glossary*. https://www.gartner.com/it-glossary/prescriptive-analytics/. Version: 2017. – Last accessed: 2018-03-29

[178] Gartner Group: *Data Integration - Gartner IT Glossary*. https://www.gartner.com/it-glossary/data-integration-tools/. Version: 2018. – Last accessed: 2018-04-12

[179] Gartner Group: *Gartner Says Self-Service Analytics and BI Users Will Produce More Analysis Than Data Scientists Will by 2019*. https://www.gartner.com/newsroom/id/3848671. Version: 2018. – Last accessed: 2018-03-30

[180] Gartner Inc.: *Business Intelligence - BI - Gartner IT Glosary*. http://www.gartner.com/it-glossary/business-intelligence-bi/. – Last accessed: 2017-08-08

[181] Gartner Inc.: *Gartner's 2011 Hype Cycle Special Report Evaluates the Maturity of 1,900 Technologies*. https://www.gartner.com/newsroom/id/1763814. Version: 2011. – Last accessed: 2018-03-15

[182] Gartner Inc.: *Gartner's 2012 Hype Cycle for Emerging Technologies Identifies "Tipping Point" Technologies That Will Unlock Long-Awaited Technology Scenarios*. https://www.gartner.com/newsroom/id/2124315. Version: 2012. – Last accessed: 2018-03-20

[183] Gartner Inc.: *Hype Cycle for In-Memory Computing Technology, 2012*. https://www.gartner.com/doc/2098815/hype-cycle-inmemory-computing-technology. Version: 2012

[184] Gartner Inc.: *What Is Big Data? - Gartner IT Glossary - Big Data*. https://www.gartner.com/it-glossary/big-data. Version: 2016. – Last accessed: 2018-03-18

[185] Gartner Inc.: *Magic Quadrant for Cloud Infrastructure as a Service, Worldwide*. https://www.gartner.com/doc/3738058/magic-quadrant-cloud-infrastructure-service. Version: 2017. – Last accessed: 2018-03-20

[186] Gartner Inc.: *Magic Quadrant for Data Integration Tools*. https://www.gartner.com/doc/3777464/magic-quadrant-data-integration-tools. Version: 2017. – Last accessed: 2018-04-21

[187] Gartner Inc.: *Top Trends in the Gartner Hype Cycle for Emerging Technologies, 2017 - Smarter With Gartner*. https://www.gartner.com/smarterwithgartner/top-trends-in-the-gartner-hype-cycle-for-emerging-technologies-2017/. Version: 2017. – Last accessed: 2018-03-05

[188] GitHub: *GitHub - nathanmarz/elephantdb: Distributed database specialized in exporting key/value data from Hadoop.* https://github.com/nathanmarz/elephantdb. Version: 2014. – Last accessed: 2018-05-06

[189] Google Scholar: *hanlding large data - between 1990 and 2000 - Google Scholar.* https://scholar.google.de/scholar?q=handling+large+data&hl=en&as_ylo=1990&as_yhi=2000. Version: 2018. – Last accessed: 2018-04-20

[190] Google Scholar: *Inmon: Building the Data Warehouse - Google Scholar.* https://scholar.google.com/scholar?cites=12212744057519089280&as_sdt=2005&sciodt=0,5&hl=en. Version: 2018. – Last accessed: 2018-03-07

[191] Grant, Trevor: *Getting started with Apache Mahout – IBM Watson Data – Medium.* https://medium.com/ibm-data-science-experience/getting-started-with-apache-mahout-fbd074f95e50. Version: 2017. – Last accessed: 2018-04-30

[192] Gross, Bill: *Google's Self Driving Car Gathers Nearly 1 GB/Sec | Bill Gross | Pulse | LinkedIn.* https://www.linkedin.com/pulse/20130502024505-9947747-google-s-self-driving-car-gathers-nearly-1-gb-per-second. Version: 2013. – Last accessed: 2018-03-19

[193] Grzywaczewski, Adam: *Training AI for Self-Driving Vehicles: the Challenge of Scale | NVIDIA Developer Blog.* https://devblogs.nvidia.com/training-self-driving-vehicles-challenge-scale/. Version: 2017. – Last accessed: 2018-04-07

[194] H2O.ai: *Algorithms — H2O 3.18.0.8 documentation.* http://docs.h2o.ai/h2o/latest-stable/h2o-docs/data-science.html. Version: 2018. – Last accessed: 2018-04-30

[195] H2O.ai: *H2O.ai.* https://www.h2o.ai/. Version: 2018. – Last accessed: 2018-04-30

[196] H2O.ai: *Overview — H2O Sparkling Water 2.3.1 documentation.* http://docs.h2o.ai/sparkling-water/2.3/latest-stable/doc/index.html. Version: 2018. – Last accessed: 2018-04-30

[197] H2O.ai: *Supported Data Sources — H2O Sparkling Water 2.3.1 documentation.* http://docs.h2o.ai/sparkling-water/2.3/latest-stable/doc/design/supported_data_sources.html. Version: 2018. – Last accessed: 2018-05-01

[198] H2O.ai: *Welcome to H2O 3 — H2O 3.18.0.8 documentation.* http://docs.h2o.ai/h2o/latest-stable/h2o-docs/welcome.html. Version: 2018. – Last accessed: 2018-04-30

[199] Haas, Kevin: *Working with Small Files in Hadoop - Part 2 | Hitachi Vantara Community*. https://community.hds.com/community/products-and-solutions/pentaho/blog/2017/12/15/working-with-small-files-in-hadoop-part-2. Version: 2017. – Last accessed: 2018-03-26

[200] Hackathorn, Richard: *The BI Watch: Real-Time to Real Value*. https://www.researchgate.net/publication/228498840_The_BI_watch_real-time_to_real-value. Version: 2004. – Last accessed: 2018-04-08

[201] Hagmann, Mike: *Guinness World Record - Largest Data Warehouse - SAP HANA*. https://blogs.saphana.com/2014/03/05/guinness-world-record-largest-data-warehouse/. Version: 2014. – Last accessed: 2018-05-04

[202] Hale, Coda: *You Can't Sacrifice Partition Tolerance | codahale.com*. https://codahale.com/you-cant-sacrifice-partition-tolerance/#errata10221010. Version: 2010. – Last accessed: 2018-04-13

[203] Hilliard, Rich: *ISO/IEC/IEEE 42010 : Architecture Descriptions*. http://www.iso-architecture.org/42010/ads/. Version: 2016. – Last accessed: 2017-10-04

[204] Hilliard, Rich: *ISO/IEC/IEEE 42010: Defining "architecture"*. http://www.iso-architecture.org/42010/defining-architecture.html. Version: 2016. – Last accessed: 2017-10-04

[205] Hitachi Vantara: *Data Integration - Kettle | Hitachi Vantara Community*. https://community.hds.com/docs/DOC-1009855. Version: 2017. – Last accessed: 2018-04-30

[206] Hitachi Vantara: *Mondrian | Hitachi Vantara Community*. https://community.hds.com/docs/DOC-1009853. Version: 2017. – Last accessed: 2018-04-30

[207] Horowitz, Eliot: *MongoDB Drops ACID | MongoDB*. https://www.mongodb.com/blog/post/multi-document-transactions-in-mongodb. Version: 2018. – Last accessed: 2018-04-13

[208] Hortonworks: *Manage Data-at-Rest and Deliver Big Data Analytics with Hortonworks Data Platform (HDP) | Hortonworks*. https://hortonworks.com/products/data-platforms/hdp/. Version: 2018. – Last accessed: 2018-04-14

[209] Howard, Philip: *The EDW is dead – Bloor Research*. https://www.bloorresearch.com/2011/06/edw-dead/. Version: 2011. – Last accessed: 2018-05-08

[210] Hurst, Nathan: *Visual Guide to NoSQL Systems - Nathan Hurst's Blog*. http://blog.nahurst.com/visual-guide-to-nosql-systems. Version: 2010. – Last accessed: 2018-04-13

[211] IBM Corp.: *IBM Knowledge Center - SQL OLAP functions*. https://www.ibm.com/support/knowledgecenter/en/SSCRW7_6.3.0/com.ibm.redbrick.doc6.3/sqlrg/sqlrg36.htm. Version: 2004. – Last accessed: 2018-02-23

[212] IBM Corp.: *IBM100 - Power 4: The First Multi-Core, 1GHz Processor*. http://www-03.ibm.com/ibm/history/ibm100/us/en/icons/power4/. Version: 2011. – Last accessed: 2018-03-16

[213] Iliff, David: *The exterior of Beauvais Cathedral in Picardy, France. - Wikipedia*. https://en.wikipedia.org/wiki/Beauvais_Cathedral#/media/File:Beauvais_Cathedral_Exterior_1,_Picardy,_France_-_Diliff.jpg. Version: 2015. – Last accessed: 2017-08-10

[214] Inland Revenue Authority of Singapore: *Common Tax Reliefs That Help Reduce The Tax Bills - IRAS*. https://www.iras.gov.sg/irashome/Businesses/Companies/Learning-the-basics-of-Corporate-Income-Tax/Common-Tax-Reliefs-That-Help-Reduce-The-Tax-Bills/#title1. Version: 2018. – Last accessed: 2018-04-16

[215] Internet Archive Book Images on Flickr: *Image from page 84 of "L'architecture. Le passé.–Le présent" (1916)| Flickr*. https://www.flickr.com/photos/126377022@N07/14778167895. – Last accessed: 2017-10-08

[216] Internetlivestats.com: *1 Second - Internet Live Stats*. http://www.internetlivestats.com/one-second/. Version: 2018. – Last accessed: 2018-03-19

[217] IT gmbh solid: *DB-Engines Ranking - popularity ranking of database management systems*. https://db-engines.com/en/ranking. Version: 2018. – Last accessed: 2018-04-13

[218] IT gmbh solid: *DB-Engines Ranking per database model category*. https://db-engines.com/en/ranking_categories. Version: 2018. – Last accessed: 2018-04-12

[219] IT gmbh solid: *Redis System Properties*. https://db-engines.com/en/system/Redis. Version: 2018. – Last accessed: 2018-04-30

[220] IT gmbh solid: *Riak KV System Properties*. https://db-engines.com/en/system/Riak+KV. Version: 2018. – Last accessed: 2018-04-30

[221] Kaggle Inc.: *House Prices: Advanced Regression Techniques | Kaggle*. https://www.kaggle.com/c/house-prices-advanced-regression-techniques/data. Version: 2016. – Last accessed: 2018-05-03

[222] Kaggle Inc.: *House Prices: Advanced Regression Techniques | Kaggle*. https://www.kaggle.com/c/house-prices-advanced-regression-techniques. Version: 2016. – Last accessed: 2018-05-02

[223] Kaggle Inc.: *Kaggle: Your Home for Data Science*. https://www.kaggle.com/. Version: 2018. – Last accessed: 2018-05-02

[224] Kawa, Adam: *Recent Evolution of Zero Data Loss Guarantee in Spark Streaming With Kafka | GetInData*. http://getindata.com/recent-evolution-of-zero-data-loss-guarantee-in-spark-streaming-with-kafka/. Version: 2015. – Last accessed: 2018-04-19

[225] Kekrse, Amol: *Apache Apex (Incubating) Introduction ' DataTorrent Blog*. https://www.datatorrent.com/blog/introducing-apache-apex-incubating/. Version: 2016. – Last accessed: 2018-04-11

[226] Khupat, Shrikant ; Ahire, Atul: *Data warehouse augmentation, Part 4: Use big data technologies for initial data exploration*. https://www.ibm.com/developerworks/analytics/library/ba-augment-data-warehouse4/ba-augment-data-warehouse4-pdf.pdf. Version: 2014. – Last accessed: 2018-04-15

[227] Kindi, Rahul: *vidDistill - Shorten YouTube Videos Using Captions*. https://viddistill.herokuapp.com/. Version: 2018. – Last accessed: 2018-05-05

[228] Kleppmann, Martin: *Please stop calling databases CP or AP — Martin Kleppmann's blog*. https://martin.kleppmann.com/2015/05/11/please-stop-calling-databases-cp-or-ap.html. Version: 2015. – Last accessed: 2018-04-13

[229] KNIME: *Seven Techniques for Data Dimensionality Reduction | KNIME*. https://www.knime.com/blog/seven-techniques-for-data-dimensionality-reduction. Version: 2015. – Last accessed: 2018-04-04

[230] KNIME AG: *Data Access | KNIME*. https://www.knime.com/nodeguide/data-access. Version: 2018. – Last accessed: 2018-05-01

[231] KNIME AG: *Features | KNIME*. https://www.knime.com/features. Version: 2018. – Last accessed: 2018-04-30

[232] KNIME AG: *KNIME - Open for Innovation*. https://www.knime.com/. Version: 2018. – Last accessed: 2018-04-30

[233] KNIME AG: *KNIME Big Data Connectors | KNIME*. https://www.knime.com/knime-big-data-connectors. Version: 2018. – Last accessed: 2018-05-01

[234] Kobielus, James: *No, the data warehouse is not dead | InfoWorld*. https://www.infoworld.com/article/2908085/big-data/no-the-data-warehouse-is-not-dead.html. Version: 2015. – Last accessed: 2018-05-08

[235] Krikorian, Raffi: *New Tweets per second record, and how!* https://blog.twitter.com/engineering/en_us/a/2013/new-tweets-per-second-record-and-how.html. Version: 2013. – Last accessed: 2018-03-19

[236] Kuang, Hairong ; Nykiel, Tom: *Running the largest HDFS cluster*. http://cloud.berkeley.edu/data/hdfs-scalability.pdfhttps://www.infoq.com/presentations/Hadoop-HDFS-Facebook. Version: 2013. – Last accessed: 2018-04-01

[237] Labs, Redis: *Redis Mass Insertion – Redis*. https://redis.io/topics/mass-insert. Version: 2018. – Last accessed: 2018-05-01

[238] LogisticTrends.com: *Analytics and the Supply Chain | Logistics Trends*. http://logisticstrends.com/weather-analytics-and-the-supply-chain/. Version: 2014. – Last accessed: 2018-03-30

[239] Lohr, Steve: *The Origins of 'Big Data': An Etymological Detective Story*. https://bits.blogs.nytimes.com/2013/02/01/the-origins-of-big-data-an-etymological-detective-story/. Version: 2013. – Last accessed: 2018-03-15

[240] Machado, Carlos: *Bluetooth beacons for tracking people and assets*. https://www.accuware.com/blog/bluetooth-beacons-tracking/. Version: 2017. – Last accessed: 2018-04-30

[241] Machine Learning Group at the University of Waikato: *Waikato Environment for Knowledge Analysis (WEKA) | WEKA Packages*. http://weka.sourceforge.net/packageMetaData/. Version: 2017. – Last accessed: 2018-04-30

[242] Machine Learning Group at the University of Waikato: *Weka 3 - Data Mining with Open Source Machine Learning Software in Java*. https://www.cs.waikato.ac.nz/ml/weka/. Version: 2018. – Last accessed: 2018-04-30

[243] MapR: *The MapR Distribution including Apache Hadoop | MapR*. https://mapr.com/products/mapr-distribution-including-apache-hadoop/. Version: 2018. – Last accessed: 2018-04-14

[244] Marr, Bernard: *A Brief History of Big Data Everyone Should Read | Bernard Marr | Pulse | LinkedIn*. https://www.linkedin.com/pulse/brief-history-big-data-everyone-should-read-bernard-marr. Version: 2015. – Last accessed: 2018-05-04

[245] Marr, Bernard: *Contradictions of Big Data - Data Science Central*. https://www.datasciencecentral.com/profiles/blogs/contradictions-of-big-data. Version: 2015. – Last accessed: 2018-03-19

[246] Marr, Bernard: *The 5 V's of Big Data by Bernard Marr | Data Science Central*. https://www.datasciencecentral.com/profiles/blogs/the-5-v-s-of-big-data-by-bernard-marr. Version: 2015. – Last accessed: 2018-03-19

[247] Marr, Bernard: *Why only one of the 5 Vs of big data really matters | IBM Big Data & Analytics Hub*. http://www.ibmbigdatahub.com/blog/why-only-one-5-vs-big-data-really-matters. Version: 2015. – Last accessed: 2018-03-20

[248] Marr, Bernard: *The Top 10 AI And Machine Learning Use Cases Everyone Should Know About*. https://www.forbes.com/sites/bernardmarr/2016/09/30/what-are-the-top-10-use-cases-for-machine-learning-and-ai/#33f5d20e94c9. Version: 2016. – Last accessed: 2018-04-07

[249] Marr, Bernard: *The Amazing Ways Google Uses Deep Learning AI*. https://www.forbes.com/sites/bernardmarr/2017/08/08/the-amazing-ways-how-google-uses-deep-learning-ai/#2a7102683204. Version: 2017. – Last accessed: 2018-04-07

[250] Martens, China: *BI at age 17 | Computerworld*. https://www.computerworld.com/article/2554088/business-intelligence/bi-at-age-17.html. Version: 2006. – Last accessed: 2018-02-14

[251] Mashey, John: *Big Data ... and the Next Wave of InfraStress*. http://static.usenix.org/event/usenix99/invited_talks/mashey.pdf. Version: 1998. – Last accessed: 2018-03-15

[252] Matz, Cade: *Google search index splits with MapReduce - The Register*. https://www.theregister.co.uk/2010/09/09/google_caffeine_explained/. Version: 2010. – Last accessed: 2018-03-27

[253] Maydon, Thomas: *The 4 Types of Data Analytics - Data Science Central*. https://www.datasciencecentral.com/profiles/blogs/the-4-types-of-data-analytics. Version: 2017. – Last accessed: 2018-03-29

[254] McCallum, John C.: *Memory Prices from 1957 to 2017*. https://jcmit.net/memoryprice.htm. Version: 2017. – Last accessed: 2018-03-16

[255] McDonald, Henry: *Ireland is cool for Google as its data servers like the weather | Technology | The Guardian*. https://www.theguardian.com/technology/2012/dec/23/ireland-cool-google-data-servers-weather. Version: 2012. – Last accessed: 2018-04-16

[256] Micro Focus International plc: *About Vertica Release Notes*. https://my.vertica.com/docs/ReleaseNotes/9.0.x/Vertica_9.0.x_Release_Notes.htm#9.0.1-7. Version: 2018. – Last accessed: 2018-04-26

[257] Micro Focus International plc: *How Projections are Created*. https://my.vertica.com/docs/9.0.x/HTML/index.htm#Authoring/ConceptsGuide/Components/HowProjectionsAreCreated.htm%3FTocPath%3DVertica%2520Concepts%7CVertica%2520Components%7CPhysical%2520Schema%7C_____3. Version: 2018. – Last accessed: 2018-04-26

[258] Micro Focus International plc: *SQL Reference Manual*. https://my.vertica.com/docs/9.0.x/HTML/index.htm#Authoring/SQLReferenceManual/SQLReferenceManual.htm%3FTocPath%3DSQL%2520Reference%2520Manual%7C_____0. Version: 2018. – Last accessed: 2018-04-26

[259] Micro Focus International plc: *Transaction*. https://my.vertica.com/docs/9.0.x/HTML/index.htm#Authoring/Glossary/Transaction.htm. Version: 2018. – Last accessed: 2018-04-26

[260] Micro Focus International plc: *Vertica Cluster Architecture*. https://my.vertica.com/docs/9.0.x/HTML/index.htm#Authoring/ConceptsGuide/Components/ArchitectureOfTheHPVerticaCluster.htm%3FTocPath%3DVertica%2520Concepts%7C_____1. Version: 2018. – Last accessed: 2018-04-26

[261] Microsoft: *GROUPING SETS Equivalents*. https://technet.microsoft.com/en-us/library/bb510427(v=sql.105).aspx. – Last accessed: 2018-02-23

[262] Microsoft: *Using GROUP BY with ROLLUP, CUBE, and GROUPING SETS*. https://technet.microsoft.com/en-us/library/bb522495(v=sql.105).aspx. – Last accessed: 2018-02-23

[263] Microsoft: *Fun with ML, Stream Analytics and PowerBI – Observing Virality in Real Time*. https://blogs.technet.microsoft.com/machinelearning/2015/05/04/fun-with-ml-stream-analytics-and-powerbi-observing-virality-in-real-time/. Version: 2015. – Last accessed: 2017-09-07

[264] Microsoft: *How old do I look?* https://how-old.net. Version: 2015. – Last accessed: 2017-09-07

[265] Microsoft: *Aggregate Functions (Transaction-SQL) | Microsoft Docs*. https://docs.microsoft.com/en-us/sql/t-sql/functions/aggregate-functions-transact-sql. Version: 2018

[266] Microsoft Corporation: *Data Mining Algorithms (Analysis Services - Data Mining) | Microsoft Docs*. https://docs.microsoft.com/en-us/sql/analysis-services/data-mining/data-mining-algorithms-analysis-services-data-mining?view=sql-analysis-services-2017. Version: 2016. – Last accessed: 2018-04-30

[267] Microsoft Corporation: *Feature Selection (Data Mining) | Microsoft Docs*. https://docs.microsoft.com/en-us/sql/analysis-services/data-mining/feature-selection-data-mining?view=sql-analysis-services-2017. Version: 2016. – Last accessed: 2018-04-30

[268] Microsoft Corporation: *Microsoft to acquire LinkedIn | Stories*. https://news.microsoft.com/2016/06/13/microsoft-to-acquire-linkedin/. Version: 2016. – Last accessed: 2018-04-16

[269] Microsoft Corporation: *High availability and Scalability in Analysis Services | Microsoft Docs*. https://docs.microsoft.com/en-us/sql/analysis-services/instances/high-availability-and-scalability-in-analysis-services?view=sql-analysis-services-2017. Version: 2017. – Last accessed: 2018-04-30

[270] Microsoft Corporation: *Supported Data Sources (SSAS - Multidimensional) | Microsoft Docs*. https://docs.microsoft.com/en-us/sql/analysis-services/multidimensional-models/supported-data-sources-ssas-multidimensional?view=sql-analysis-services-2017. Version: 2017. – Last accessed: 2018-04-30

[271] Microsoft Corporation: *About SQL Server Analysis Services | Microsoft Docs*. https://docs.microsoft.com/en-us/sql/analysis-services/analysis-services?view=sql-analysis-services-2017. Version: 2018. – Last accessed: 2018-04-30

[272] Mierswa, Ingo: *Time Series Forecasting with RapidMiner and R | RapidMiner*. https://rapidminer.com/blog/time-series-forecasting-rapidminer-r/. Version: 2017. – Last accessed: 2018-04-30

[273] Miller, Ron: *SAP snags CallidusCloud for $2.4 billion | TechCrunch*. https://techcrunch.com/2018/01/30/sap-snags-calliduscloud-for-2-4-billion/. Version: 2018. – Last accessed: 2018-04-13

[274] Mnasri, Maali: *Quick review on Text Clustering and Text Similarity Approaches - LumenAI*. http://www.lumenai.fr/blog/quick-review-on-text-clustering-and-text-similarity-approaches. Version: 2016. – Last accessed: 2018-05-08

[275] Monash, Curt: *December 2, 2007 Disputed history of the term Business Intelligence | Software Memories*. http://www.softwarememories.com/2007/12/02/disputed-history-of-the-term-business-intelligence/. Version: 2007. – Last accessed: 2018-02-14

[276] MongoDB Inc.: *Aggregation — MongoDB Manual 3.6*. https://docs.mongodb.com/manual/aggregation/#aggregation-framework. Version: 2018. – Last accessed: 2018-04-30

[277] MongoDB Inc.: *Aggregation Pipeline Quick Reference — MongoDB Manual 3.6*. https://docs.mongodb.com/manual/meta/aggregation-quick-reference/#aggregation-accumulator-operators. Version: 2018. – Last accessed: 2018-04-30

[278] MongoDB Inc.: *Atomicity and Transactions — MongoDB Manual 3.6*. https://docs.mongodb.com/manual/core/write-operations-atomicity/. Version: 2018. – Last accessed: 2018-04-13

[279] MongoDB Inc.: *$lookup (aggregation) — MongoDB Manual 3.6*. https://docs.mongodb.com/manual/reference/operator/aggregation/lookup/. Version: 2018. – Last accessed: 2018-04-30

[280] MongoDB Inc.: *MongoDB CRUD Operations — MongoDB Manual 3.6*. https://docs.mongodb.com/manual/crud/. Version: 2018. – Last accessed: 2018-04-30

[281] MongoDB Inc.: *MongoDB for GIANT Ideas | MongoDB*. https://www.mongodb.com/. Version: 2018. – Last accessed: 2018-04-30

[282] MongoDB Inc.: *MongoDB Limits and Thresholds — MongoDB Manual 3.6*. https://docs.mongodb.com/manual/reference/limits/. Version: 2018. – Last accessed: 2018-04-24

[283] MongoDB Inc.: *The MongoDB 3.6 Manual — MongoDB Manual 3.6*. https://docs.mongodb.com/manual/. Version: 2018. – Last accessed: 2018-04-13

[284] Msv, Janakiram: *All the Apache Streaming Projects: An Exploratory Guide - The New Stack*. https://thenewstack.io/apache-streaming-projects-exploratory-guide/. Version: 2016. – Last accessed: 2018-04-12

[285] Murthy, Arun: *Apache Hadoop YARN - Concepts and Applications - Hortonworks*. hortonworks.com/blog/apache-hadoop-yarn-concepts-and-applications/. Version: 2012. – Last accessed: 2018-03-28

[286] Narkhede, Neha: *Exactly-once Semantics is Possible: Here's How Apache Kafka Does it*. https://www.confluent.io/blog/exactly-once-semantics-are-possible-heres-how-apache-kafka-does-it/. Version: 2017. – Last accessed: 2018-04-09

[287] Narsude, Chetan: *Real-Time Event Stream Processing - Paradigms and Low Latency*. https://www.datatorrent.com/blog/real-time-event-stream-processing-what-are-your-choices/. Version: 2015. – Last accessed: 2018-04-09

[288] Nelson, John P.: *MicroGnu for MSDOS*. https://groups.google.com/forum/#!original/comp.sources.misc/d3EXP4D_VK8/x7WrVBMb5FgJ. Version: 1987. – Last accessed: 2018-03-15

[289] Neo4j Inc.: *3.5.1. Indexes - 3.5. Schema*. https://neo4j.com/docs/developer-manual/current/cypher/schema/index/. Version: 2018. – Last accessed: 2018-04-30

[290] Neo4j Inc.: *For Relational Database Developers: A SQL to Cypher Guide*. https://neo4j.com/developer/guide-sql-to-cypher/. Version: 2018. – Last accessed: 2018-04-30

[291] Neo4j Inc.: *Neo4j : Export a (sub)graph to Cypher script and import it again - Neo4j Graph Database Platform*. https://neo4j.com/developer/kb/export-subgraph-to-cypher-and-import/. Version: 2018. – Last accessed: 2018-05-01

[292] Neo4j Inc.: *Neo4j Cypher Refcard 3.3*. https://neo4j.com/docs/cypher-refcard/current/. Version: 2018. – Last accessed: 2018-04-30

[293] Neo4j Inc.: *Neo4j's Graph Query Language: An Introduction to Cypher*. https://neo4j.com/developer/cypher-query-language/. Version: 2018. – Last accessed: 2018-04-30

[294] Neo4j Inc.: *Release Notes Archive - Neo4j Graph Database Platform*. https://neo4j.com/release-notes/. Version: 2018. – Last accessed: 2018-04-30

[295] Neo4j Inc.: *The Neo4j Graph Platform – The #1 Platform for Connected Data*. https://neo4j.com/. Version: 2018. – Last accessed: 2018-04-30

[296] Neo4j Staff: *Recap: Intro to Graph Databases | Webinar Series #1 - Neo4j Graph Database Platform*. https://neo4j.com/blog/recap-intro-to-graph-databases-webinar-series-1/. Version: 2011. – Last accessed: 2018-04-14

[297] O'Dell, Kevin: *How-to: Select the Right Hardware for Your New Hadoop Cluster - Cloudera Engineering Blog*. http://blog.cloudera.com/blog/2013/08/how-to-select-the-right-hardware-for-your-new-hadoop-cluster/. Version: 2013. – Last accessed: 2018-03-27

[298] Oracle Corporation: *MySQL :: MySQL 8.0 Reference Manual :: A.10 MySQL 8.0 FAQ: MySQL Cluster*. https://dev.mysql.com/doc/refman/8.0/en/faqs-mysql-cluster.html. – Last accessed: 2018-04-30

[299] Oracle Corporation: *Big Data Products - Big Data Analytics| Oracle*. https://www.oracle.com/big-data/products.html#manage. Version: 2014. – Last accessed: 2018-03-29

[300] Oracle Corporation: *Aggregate Functions*. https://docs.oracle.com/cloud/latest/db112/SQLRF/functions003.htm#SQLRF20035. Version: 2018. – Last accessed: 2018-03-05

[301] Oracle Corporation: *MySQL*. https://www.mysql.com/. Version: 2018. – Last accessed: 2018-04-30

[302] Oracle Corporation: *MySQL :: MySQL 5.7 Reference Manual :: 14.5.2.2 autocommit, Commit, and Rollback*. https://dev.mysql.com/doc/refman/5.7/en/innodb-autocommit-commit-rollback.html. Version: 2018. – Last accessed: 2018-01-18

[303] Oracle Corporation: *MySQL :: MySQL 5.7 Reference Manual :: 15.3 The MEMORY Storage Engine*. https://dev.mysql.com/doc/refman/5.7/en/memory-storage-engine.html. Version: 2018. – Last accessed: 2018-04-16

[304] Oracle Corporation: *MySQL :: MySQL 5.7 Reference Manual :: 1.8 MySQL Standards Compliance*. https://dev.mysql.com/doc/refman/5.7/en/compatibility.html. Version: 2018. – Last accessed: 2018-01-19

[305] Oracle Corporation: *MySQL :: MySQL 5.7 Reference Manual :: 21 MySQL NDB Cluster 7.5 and NDB Cluster 7.6*. https://dev.mysql.com/doc/refman/5.7/en/mysql-cluster.html. Version: 2018. – Last accessed: 2018-04-30

[306] Oracle Corporation: *MySQL :: MySQL 5.7 Reference Manual :: 21.1.5.1 Differences Between the NDB and InnoDB Storage Engines*. https://dev.mysql.com/doc/refman/5.7/en/mysql-cluster-ndb-innodb-engines.html. Version: 2018. – Last accessed: 2018-04-30

[307] Oracle Corporation: *MySQL :: MySQL 5.7 Reference Manual :: 21.1.6 Known Limitations of NDB Cluster*. https://dev.mysql.com/doc/refman/5.7/en/mysql-cluster-limitations.html. Version: 2018. – Last accessed: 2018-04-30

[308] Oracle Corporation: *MySQL :: MySQL 8.0 Reference Manual :: 12.19.2 GROUP BY Modifiers*. https://dev.mysql.com/doc/refman/8.0/en/group-by-modifiers.html. Version: 2018. – Last accessed: 2018-04-30

[309] Oracle Corporation: *MySQL :: MySQL 8.0 Reference Manual :: 12.20 Window Functions*. https://dev.mysql.com/doc/refman/8.0/en/window-functions.html. Version: 2018. – Last accessed: 2018-04-30

[310] Oracle Corporation: *MySQL :: MySQL 8.0 Reference Manual :: 1.4 What Is New in MySQL 8.0*. https://dev.mysql.com/doc/refman/8.0/en/mysql-nutshell.html. Version: 2018. – Last accessed: 2018-04-30

[311] Oracle Corporation: *MySQL :: MySQL 8.0 Release Notes*. https://dev.mysql.com/doc/relnotes/mysql/8.0/en/. Version: 2018. – Last accessed: 2018-04-30

[312] Oracle Corporation: *MySQL :: MySQL NDB Cluster 7.5 Release Notes*. https://dev.mysql.com/doc/relnotes/mysql-cluster/7.5/en/. Version: 2018. – Last accessed: 2018-04-30

[313] Oracle Corporation: *MySQL :: MySQL NDB Cluster API Developer Guide*. https://dev.mysql.com/doc/ndbapi/en/. Version: 2018. – Last accessed: 2018-04-30

[314] Oracle Corporation: *MySQL :: MySQL Workbench*. https://www.mysql.com/de/products/workbench/. Version: 2018. – Last accessed: 2018-02-26

[315] Oracle Corporation: *Oracle Database 18*. http://www.oracle.com/technetwork/database/enterprise-edition/overview/index.html. Version: 2018. – Last accessed: 2018-03-29

[316] Oracle Corporation: *Oracle TimesTen In-Memory Database Product Center*. http://www.oracle.com/technetwork/database/database-technologies/timesten/overview/index.html. Version: 2018. – Last accessed: 2018-03-29

[317] OSRAM: Technical Application Guide - EINSTONE module. Version: 2017. https://media.osram.info/im/img/osram-dam-2488767/downloads/App_Guide_EINSTONE_module_(EN).pdf. Munich, Germany, 2017. – Technical report. – 8 P. – ISBN 1530752752

[318] Oxford University Press: *architecture | Definition of architecture in English by Oxford Dictionaries*. https://en.oxforddictionaries.com/definition/architecture. Version: 2018. – Last accessed: 2018-04-17

[319] Oxford University Press: *information explosion - Definition of information explosion in English by Oxford Dictionaries*. https://en.oxforddictionaries.com/definition/information_explosion. Version: 2018. – Last accessed: 2018-03-15

[320] Oxford University Press: *intelligence | Definition of intelligence in English by Oxford Dictionaries*. https://en.oxforddictionaries.com/definition/intelligence. Version: 2018. – Last accessed: 2018-04-19

[321] Oxford University Press: *paradigm | Definition of paradigm in English by Oxford Dictionaries*. https://en.oxforddictionaries.com/definition/paradigm. Version: 2018. – Last accessed: 2018-04-05

[322] Patterson, Pat: *Making Sense of Stream Processing - StreamSets*. https://streamsets.com/blog/making-sense-stream-processing/. Version: 2017. – Last accessed: 2018-04-12

[323] Pentaho: *Pentaho Mondrian Documentation*. https://mondrian.pentaho.com/documentation/mdx.php. Version: 2011. – Last accessed: 2018-04-30

[324] Piatesky, Gregory: *New Leader, Trends, and Surprises in Analytics, Data Science, Machine Learning Software Poll*. https://www.kdnuggets.com/2017/05/poll-analytics-data-science-machine-learning-software-leaders.html/2. Version: 2017. – Last accessed: 2018-04-21

[325] Piatestky, Gregory: *Gartner 2017 Magic Quadrant for Data Science Platforms: gainers and losers*. https://www.kdnuggets.com/2017/02/gartner-2017-mq-data-science-platforms-gainers-losers.html. Version: 2017. – Last accessed: 2018-05-06

[326] Plante, Amber: *Informed Decisions with Predictive Analytics | The Weather Company*. https://business.weather.com/blog/going-beyond-weather-forecasts-informed-decision-making-with-predictive-analytics. Version: 2016. – Last accessed: 2018-03-30

[327] Popescu, Alina: *Analysts break down Oracle's new cloud business model | #OOW14 - SiliconANGLE*. https://siliconangle.com/blog/2014/10/01/oracles-new-business-model-is-based-on-cloud-and-converged-infrastructure-oow14/. Version: 2014. – Last accessed: 2018-03-29

[328] Portilla, Jose: *A Beginner's Guide to Neural Networks in Python and SciKit Learn 0.18 | Springboard Blog*. https://www.springboard.com/blog/beginners-guide-neural-network-in-python-scikit-learn-0-18/. Version: 2017. – Last accessed: 2018-05-05

[329] Poss, Raphael: *Modesty in Simplicity: CockroachDB's JOIN | Cockroach Labs*. https://www.cockroachlabs.com/blog/cockroachdbs-first-join/. Version: 2016. – Last accessed: 2018-04-30

[330] PostgreSQL Global Development Group: *PostgreSQL: Documentation: 10: Appendix D. SQL Conformance*. https://www.postgresql.org/docs/10/static/features.html. Version: 2018. – Last accessed: 2018-01-19

[331] PostgreSQL Global Development Group: *PostgreSQL: Documentation: 10: D.1. Supported Features*. https://www.postgresql.org/docs/current/static/features-sql-standard.html. Version: 2018. – Last accessed: 2018-04-30

[332] PostgreSQL Global Development Group: *PostgreSQL: Documentation: 10: D.2. Unsupported Features*. https://www.postgresql.org/docs/current/static/unsupported-features-sql-standard.html. Version: 2018. – Last accessed: 2018-04-30

[333] PostgreSQL Global Development Group: *PostgreSQL: The world's most advanced open source database*. https://www.postgresql.org/. Version: 2018. – Last accessed: 2018-04-30

[334] Power, D. J.: *A Brief History of Decision Support Systems*. http://dssresources.com/history/dsshistory.html. Version: 2007. – Last accessed: 2018-02-14

[335] Power, D. J.: *Ask Dan! about DSS - Who originated the term Business Intelligence (BI)?* http://dssresources.com/faq/index.php?action=artikel&id=360. Version: 2016. – Last accessed: 2018-02-14

[336] Quinn, Matt: *Forget big data – let's talk about all data | VentureBeat*. https://venturebeat.com/2013/09/03/forget-big-data-lets-talk-about-all-data/. Version: 2013. – Last accessed: 2018-03-18

[337] RapidMiner GmbH: *Configuring Radoop Connections - RapidMiner Documentation*. https://docs.rapidminer.com/latest/radoop/installation/configuring-radoop-connections/. Version: 2018. – Last accessed: 2018-05-01

[338] RapidMiner GmbH: *Data Loading Guide - RapidMiner Documentation*. https://docs.rapidminer.com/latest/radoop/overview/import-guide.html. Version: 2018. – Last accessed: 2018-05-01

[339] RapidMiner GmbH: *Operator Manual - RapidMiner Documentation*. https://docs.rapidminer.com/latest/studio/operators/. Version: 2018. – Last accessed: 2018-04-30

[340] RapidMiner GmbH: *Visual Workflow Designer for Data Scientists | RapidMiner Studio | RapidMiner*. https://rapidminer.com/products/studio/. Version: 2018. – Last accessed: 2018-04-30

[341] RapidMiner Inc.: *Code Optional Data Science for Hadoop and Spark | RapidMiner Radoop*. https://rapidminer.com/products/radoop/. Version: 2018. – Last accessed: 2018-04-20

[342] Ravindra, Savaram: *The Machine Learning Algorithms Used in Self-Driving Cars*. https://www.kdnuggets.com/2017/06/machine-learning-algorithms-used-self-driving-cars.html. Version: 2017. – Last accessed: 2018-04-07

[343] Redis Lab: *Redis Cluster Specification – Redis*. https://redis.io/topics/cluster-spec. Version: 2018. – Last accessed: 2018-04-14

[344] Redis Labs: *An introduction to Redis data types and abstractions – Redis*. https://redis.io/topics/data-types-intro. Version: 2018. – Last accessed: 2018-04-14

[345] Redis Labs: *Command reference – Redis*. https://redis.io/commands. Version: 2018. – Last accessed: 2018-04-14

[346] Redis Labs: *Data types – Redis*. https://redis.io/topics/data-types. Version: 2018. – Last accessed: 2018-04-24

[347] Redis Labs: *DUMP – Redis*. https://redis.io/commands/dump. Version: 2018. – Last accessed: 2018-05-01

[348] Redis Labs: *FAQ – Redis*. https://redis.io/topics/faq. Version: 2018. – Last accessed: 2018-04-14

[349] Redis Labs: *Redis*. https://redis.io/documentation. Version: 2018. – Last accessed: 2018-04-30

[350] Redis Labs: *Redis*. https://redis.io/. Version: 2018. – Last accessed: 2018-04-30

[351] Redis Labs: *Redis cluster tutorial – Redis*. https://redis.io/topics/cluster-tutorial. Version: 2018. – Last accessed: 2018-04-14

[352] Redis Labs / GitHub: *Redis 3.0 Release Notes - GitHub*. https://raw.githubusercontent.com/antirez/redis/3.0/00-RELEASENOTES. Version: 2016. – Last accessed: 2018-05-06

[353] Richardson, V. J.: *STEPE—Social, Technical, Economic, Political and Ecological Factor Model*. https://pages.gseis.ucla.edu/faculty/richardson/STEPE.htm. Version: 2017. – Last accessed: 2018-04-16

[354] Roman, Javi: *The Hadoop Ecosystem Table*. https://hadoopecosystemtable.github.io/. Version: 2017. – Last accessed: 2018-04-14

[355] Saini, Birender: *NiFi vs Falcon vs Oozie | Birender Saini | Pulse | LinkedIn*. https://www.linkedin.com/pulse/nifi-vs-falcon-oozie-birender-saini. Version: 2015. – Last accessed: 2018-04-15

[356] Sanders, Dale: *Healthcare Business Intelligence: What Your Strategy Needs*. https://www.healthcatalyst.com/healthcare-business-intelligence-data-warehouse. Version: 2013. – Last accessed: 2018-05-04

[357] SAP SE: *Pricing and Packaging | SAP Cloud Platform*. https://cloudplatform.sap.com/pricing.html. Version: 2018. – Last accessed: 2018-05-09

[358] Sato, Kaz: *AutoML Vision in action: from ramen to branded goods | Google Cloud Big Data and Machine Learning Blog | Google Cloud*. https://cloud.google.com/blog/big-data/2018/03/automl-vision-in-action-from-ramen-to-branded-goods. Version: 2018. – Last accessed: 2018-04-04

[359] Scott, Jimm: *A tale of two clusters: Mesos and YARN - O'Reilly Media*. https://www.oreilly.com/ideas/a-tale-of-two-clusters-mesos-and-yarn. Version: 2015. – Last accessed: 2018-04-15

[360] Shweta, Jain ; Nandi, Sujay: *Data warehouse augmentation, Part 2: Use big data technologies as a landing zone for source data*. https://www.ibm.com/developerworks/analytics/library/ba-augment-data-warehouse2/index.html?ca=drs-. Version: 2014. – Last accessed: 2018-04-15

[361] Sicilani, Toni: *Big Data Ingestion: Flume, Kafka, and NiFi - DZone Big Data*. https://dzone.com/articles/big-data-ingestion-flume-kafka-and-nifi. Version: 2017. – Last accessed: 2018-04-30

[362] Sing, Sandeep: *MapReduce Input Split versus HDFS Blocks - The Hadoop and Big Data Journal Skip to content*. https://hadoopjournal.wordpress.com/2015/06/30/mapreduce-input-split-versus-hdfs-blocks/. Version: 2015. – Last accessed: 2018-03-27

[363] Smith, Tony: *Intel dual-core Smithfield to ship as Pentium D - The Register*. https://www.theregister.co.uk/2005/03/01/intel_pentium_d/. Version: 2005. – Last accessed: 2018-03-16

[364] Smyth, Jeremy: *Converting InnoDB Tables to MySQL Cluster | Oracle Learning MySQL Blog*. https://blogs.oracle.com/jsmyth/converting-innodb-tables-to-mysql-cluster. Version: 2013. – Last accessed: 2018-05-01

[365] SPIEGEL Online: *SPIEGEL: Neue ICE-Modelle haben Probleme mit Steuerungssoftware - SPIEGEL ONLINE*. http://www.spiegel.de/wirtschaft/unternehmen/spiegel-neue-ice-modelle-haben-probleme-mit-steuerungssoftware-a-869137.html. Version: 2012. – Last accessed: 2018-04-08

[366] Srinivasan, Manikandan: *How to do Joins in Apache Cassandra™ and DataStax Enterprise | DataStax*. https://www.datastax.com/2015/03/how-to-do-joins-in-apache-cassandra-and-datastax-enterprise. Version: 2015. – Last accessed: 2018-04-14

[367] StatCounter: *Mobile Operating System Market Share Worldwide | StatCounter Global Stats*. http://gs.statcounter.com/os-market-share/mobile/worldwide. Version: 2018. – Last accessed: 2018-04-17

[368] Stibel, Jeff: *Google's Secret Weapon: MapReduce*. https://hbr.org/2008/12/mapreduce-googles-secret-weapo. Version: 2008. – Last accessed: 2018-03-27

[369] Stonebraker, Michael: *Why Enterprises Are Uninterested in Nosql | blog@CACM | Communications of ACM*. https://m.cacm.acm.org/blogs/blog-cacm/99512-why-enterprises-are-uninterested-in-nosql/fulltext. Version: 2010. – Last accessed: 2018-03-05

[370] Strozzi, Carlo: *NoSQL Relational Database Management System: Home Page*. http://www.strozzi.it/cgi-bin/CSA/tw7/I/en_US/nosql/HomePage. Version: 2010. – Last accessed: 2018-04-13

[371] Swoyer, Steve: *It's the End of the Data Warehouse as We Know It | Transforming Data with Intelligence*. https://tdwi.org/articles/2017/01/11/end-of-the-data-warehouse-as-we-know-it.aspx. Version: 2017. – Last accessed: 2018-05-04

[372] Talend: *Big Data components — Talend Components Reference Guide — Welcome to Talend Help Center*. https://help.talend.com/reader/KxVIhxtXBBFymmkkWJ{~}O4Q/MDauhq9nl1m0FXZX{~}aHp9Q. Version: 2018. – Last accessed: 2018-04-30

[373] Talend: *Open Source Big Data Tool: Talend Open Studio Free Big Data*. https://www.talend.com/products/big-data/big-data-open-studio/. Version: 2018. – Last accessed: 2018-04-30

[374] Talend: *Open Source ETL and Free Data Integration: Talend Open Studio*. https://www.talend.com/products/talend-open-studio/. Version: 2018. – Last accessed: 2018-04-30

[375] Talend: *Talend - Components and Connectors - Compatibility Matrix*. https://www.talendforge.org/components/index.php. Version: 2018. – Last accessed: 2018-04-30

[376] Talend: *Talend Data Integration Connectors, Systems & Specifications*. https://www.talend.com/products/specifications-data-integration/. Version: 2018. – Last accessed: 2018-04-30

[377] Tawakol, Omar: *The Death of the Data Warehouse and the Age of Activation | HuffPost*. https://www.huffingtonpost.com/omar-tawakol/the-death-of-the-data-warehouse_b_1669262.html. Version: 2012. – Last accessed: 2018-05-08

[378] The Open Group: *The ArchiMate® Enterprise Architecture Modeling Language | The Open Group*. http://www.opengroup.org/subjectareas/enterprise/archimate-overview. Version: 2016. – Last accessed: 2018-03-20

[379] Thompson, Nicolas ; Vogelstein, Fred: *Inside Facebook's Hellish Two Years – and Mark Zuckerberg's Struggle to Fix it All | WIRED*. https://www.wired.com/story/inside-facebook-mark-zuckerberg-2-years-of-hell. Version: 2018. – Last accessed: 2018-03-19

[380] Turck, Matt: *Is Big Data Still a Thing? (The 2016 Big Data Landscape)*. http://mattturck.com/big-data-landscape/. Version: 2016. – Last accessed: 2018-03-20

[381] Turck, Matt: *Firing on All Cylinders: The 2017 Big Data Landscape*. http://mattturck.com/bigdata2017/. Version: 2017. – Last accessed: 2018-03-20

[382] Turek, Rasty: *How many terabytes of storage does Youtube (Google) currently have? - Quora*. https://www.quora.com/How-many-terabytes-of-storage-does-Youtube-Google-currently-have. Version: 2015. – Last accessed: 2018-03-16

[383] Turnitin LLC: *FAQs | Plagiarism Software | iThenticate*. http://www.ithenticate.com/products/faqs. Version: 2017. – Last accessed: 2018-05-04

[384] Turnitin LLC: *Plagiarism Detection Comparison Database | iThenticate*. http://www.ithenticate.com/content. Version: 2017. – Last accessed: 2018-05-04

[385] Turnitin LLC: *Plagiarism Detection Software | iThenticate*. http://www.ithenticate.com/. Version: 2017. – Last accessed: 2018-05-04

[386] Turnitin LLC: *Plagiarism Detection Software Misconceptions*. http://www.ithenticate.com/resources/papers/plagiarism-detection-software-misconceptions. Version: 2017. – Last accessed: 2018-05-04

[387] Twitter: *PowerTrack API - Twitter Developers*. https://developer.twitter.com/en/docs/tweets/filter-realtime/overview/powertrack-api. Version: 2018. – Last accessed: 2018-05-03

[388] Twitter: *Standard search API - Twitter Developers*. https://developer.twitter.com/en/docs/tweets/search/api-reference/get-search-tweets. Version: 2018. – Last accessed: 2018-04-23

[389] University of Waikato: *Weka 3 - Mining Big Data with Open Source Machine Learning Software in Java*. https://www.cs.waikato.ac.nz/ml/weka/bigdata.html. Version: 2017. – Last accessed: 2018-04-20

[390] US Department of Commerce, NOAA, National Weather S.: *About Supercomputers*. https://www.weather.gov/about/supercomputers. Version: 2017. – Last accessed: 2018-03-30

[391] US Dept. of Defence / Office of the DoD CIO: Reference Architecture Description. Version: 2010. http://dodcio.defense.gov/Portals/0/Documents/DIEA/Ref_Archi_Description_Final_v1_18Jun10.pdf. 2010 (June). – Technical report. – 22 P.

[392] Vagata, Pamela ; Wilfong, Kevin: *Scaling the Facebook data warehouse to 300 PB | Engineering Blog | Facebook Code | Facebook*. https://code.facebook.com/posts/229861827208629/scaling-the-facebook-data-warehouse-to-300-pb/. Version: 2014. – Last accessed: 2018-03-16

[393] Valve: *Steam Blog :: Making Helpful User Reviews More Helpful*. https://steamcommunity.com/games/593110/announcements/detail/2666556941788470579. Version: 2017. – Last accessed: 2018-04-11

[394] Van Gundy, Kevin: *Infographic: Understanding Scalability with Neo4j - Neo4j Graph Database Platform*. https://neo4j.com/blog/neo4j-scalability-infographic/. Version: 2015. – Last accessed: 2018-04-30

[395] Vavilapalli, Vinod K.: *Delivering on Hadoop .Next: Benchmarking Performance - Hortonworks*. https://de.hortonworks.com/blog/delivering-on-hadoop-next-benchmarking-performance/. Version: 2012. – Last accessed: 2018-03-27

[396] VidasSoft Systems Inc.: *SSAS News*. http://www.ssas-info.com/analysis-services-news?limitstart=0. Version: 2018. – Last accessed: 2018-04-30

[397] Web Science Trust: *Real-time Twitter Visualisations for the US 2016 Presidential Elections - Web Science Trust*. http://www.webscience.org/2016/11/18/real-time-twitter-visualisations-us-2016-presidential-elections/. Version: 2016. – Last accessed: 2018-05-05

[398] Weise, Thomas: *End-to-end Exactly-Once with Apache Apex - DataTorrent*. https://www.datatorrent.com/blog/end-to-end-exactly-once-with-apache-apex/. Version: 2016. – Last accessed: 2018-05-02

[399] Weise, Thomas: *One Year Anniversary of Apache Apex*. http://www.atrato.io/blog/2017/04/25/one-year-apex/. Version: 2017. – Last accessed: 2018-04-11

[400] Wojewoda, Stéphane ; Hastie, Shane: *Standish Group 2015 Chaos Report - Q&A with Jennifer Lynch*. https://www.infoq.com/articles/standish-chaos-2015. Version: 2015. – Last accessed: 2018-05-02

[401] Wood Squier, Laura E.: *Hype Cycle History on Predictive Analytics - Data Science Central*. https://www.datasciencecentral.com/forum/topics/hype-cycle-history-on-predictive-analytics. Version: 2017. – Last accessed: 2018-03-29

[402] Woodie, Alex: *Why Gartner Dropped Big Data Off the Hype Curve*. https://www.datanami.com/2015/08/26/why-gartner-dropped-big-data-off-the-hype-curve/. Version: 2015. – Last accessed: 2018-03-20

[403] Woollacott, Emma: *Amazon's Fake Review Problem Is Now Worse Than Ever, Study Suggests*. https://www.forbes.com/sites/emmawoollacott/2017/09/09/exclusive-amazons-fake-review-problem-is-now-worse-than-ever. Version: 2017. – Last accessed: 2018-03-19

[404] ZH Solutions: *ZH Solutions Indoor Positioning | ZH Solutions*. http://www.zhs.io/indoor-positioning/. Version: 2016. – Last accessed: 2017-10-25

List of Abbreviations

2PL	Two-Phase Locking.
ABB	Architecture Building Block.
ACID	Atomicity, Consistency, Isolation, and Durability.
AI	Artificial Intelligence.
API	Application Programming Interface.
BASE	Basically Available, Soft state, Eventually consistent.
BI	Business Intelligence.
BIDE	BI data entity.
BLE	Bluetooth Low Energy.
BLOB	Binary Large Object.
BPMN	Business Process Modeling Notation.
CAP	Consistency, Availability, Partition tolerance.
CEP	Complex Event Processing.
CPU	Central Processing Unit.
CQL	Cassandra Query Language.
CRM	Customer Relationship Management.

CRUD Create, Read, Update, and Delete.

CSM Customer Service Monitor.

DAG Directed Acyclic Graph.

DB Database.

DBaaS Database-as-a-Service.

DBMS Database Management System.

DBS Database System.

DDL Data Definition Language.

DML Data Manipulation Language.

DSRM Design Science Research Methodology.

DSS Decision Support System.

DWH Data Warehouse.

EA Enterprise Architecture.

EIS Executive Information System.

ELT Extract-Load-Transform.

EPC Event-driven Process Chain.

ERP Enterprise Resource Planning.

ETL Extract-Transform-Load.

ETLT Extract-Transform-Load-Transform.

FD functional dependency.

FTP File Transfer Protocol.

GOBIA Goal-oriented Business Intelligence Architectures.

GoM Guidelines of Modeling.

GPU Graphics Processing Unit.

GUI Graphical User Interface.

HDFS Hadoop Distributed File System.

HTTP Hypertext Transfer Protocol.

IaaS Infrastructure-as-a-Service.

IEC International Electrotechnical Commission.

IEEE Institute of Electrical and Electronics Engineers.

IMDB In-memory Database.

IoT Internet of Things.

IP Internet Protocol.

IS Information Systems.

ISO International Organization for Standardization.

IT Information Technology.

JDBC Java Database Connectivity.

JSON JavaScript Object Notation.

JVM Java Virtual Machine.

KDD Knowledge discovery in databases.

KPI Key Performance Indicator.

MIS Management Information System.

MOLAP Multidimensional OLAP.

MPP Massive Parallel Processing.

MRI Magnetic Resonance Imaging.

MSE Mean Square Error.

MVCC Multiversion Concurrency Control.

NAS Network Attached Storage.

NBDRA NIST Big Data Reference Architecture.

NFR non-functional requirement.

NIST National Institute of Standards and Technology.

NLP Natural Language Processing.

NoSQL Not only SQL.

ODBC Open Database Connectivity.

OLAP Online Analytical Processing.

OLTP Online Transaction Processing.

OS Operating System.

PaaS Platform-as-a-Service.

PCA Principal Component Analysis.

PESTEL Political, Economic, Socio-cultural, Technological, Environment, and Legal.

PMML Predictive Model Markup Language.

PoE Power over Ethernet.

POSIX Portable Operating System Interface.

RAM Random-access Memory.

RDBMS Relational Database Management System.

RDD Resilient Distributed Dataset.

RDF Resource Description Framework.

RE Requirements Engineering.

REST Representational State Transfer.

ROLAP Relational OLAP.

SaaS Software-as-a-Service.

SAMOA Scalable Advanced Massive Online Analysis.

SAN Storage Area Network.

SDN Software Defined Networks.

SISP Strategic Information Systems Planning.

SPE Stream Processing Engine.

SPT Stream Processing Technology.

SQL Structured Query Language.

SSAS SQL Server Analysis Services.

SSD Solid-state Disk.

TF-IDF Term Frequency–Inverse Document Frequency.

TORE Technological, Organizational, Regulatory, and Economic.

UDF User-defined Function.

UML Unified Modeling Language.

URI Uniform Resource Indicator.

VM Virtual Machine.

XML Extensible Markup Language.

YARN Yet Another Resource Negotiator.

A Architecture Literature Search Details

This appendix details the literature search on Big Data architectures, which was conducted to identify illustrative exemplars of Big Data reference architectures and uses of Big Data architectures in practice in Section 3.4 and Section 3.5 in Chapter 3. These should motivate the need for a unified reference architecture, a development process, and illustrate the diversity of architectural patterns in these architectures.

This overview of Big Data architectures is achieved by means of a literature search. It focuses on various Big Data reference architectures and other novel analytical architectures, as traditional analytical architectures and their advancements are reasonable. While a structured review is not in the scope of this work, the search conducted here is oriented at established guidelines for literature search processes with respect to outlining the search's purpose, keywords, and results documentation (cf. [VSN+09]).

After outlining the purpose of the search, the literature search process is documented. The initial search was conducted in March 2017 and focused on academic search engines. The following search terms were used to identify suitable literature for architecture examination:

The search keyword "Big Data Architecture" is chosen, as the discussion of novel technologies is often lead under the term "Big Data". Thus, the keyword "Big Data Architecture" is expected to lead to architectures that incorporate at least these technologies.

To be included in the to-be-analyzed set, a search result had to contain a textual or visual reference to the concept of a Big Data architecture. This includes, e. g., reference architecture or specific architecture implementation,

Table A.1: Search keywords and engines for literature search on Big Data architectures.

Search keyword	Search engine(s)	Search scope and remarks
Big Data Architecture	Google Scholar	Primary search site
	EBSCO Host	Evaluated first two pages
	ArXiv.org	Evaluated first page

which are the search results of interest. Results that do not match these criteria are excluded during search. However, several other types of results such as technological optimizations of architecture or discussions about architectures are also included.

Therefore, the following categories of architecture descriptions can be identified from the results:

Reference Architectures have template character and should help to construct custom architectures. They are described and analyzed in Section 3.4. In this case, six of seven results were used, as these provided sufficient data for the analysis and due to the fact that [SCG+17] is a domain-specific reference architecture.

Architecture Implementations exhibit uses of established or new technologies to construct a custom architecture. To analyze them, they are described and categorized using the proposed unified Big Data value chain to extract usage patterns. The three illustrative samples out of four can be found in Section 3.5.

Architecture Components and Technologies depict technologies or other technical components and parts of an architecture suitable for analytics and Big Data, e. g., new or enhanced storage or processing technologies. As the technical background for Big Data technologies is already established in Section 2.2 and the results do not depict entirely new categories of technologies, the results from this category are not in the scope of this analysis.

Table A.2: Relevant results of the literature search on Big Data architectures.

Publication	Category	Used	Source
[PP15]	Reference Architectures	✔	Search keyword
[NIS15b]	Reference Architectures	✔	Reference of (Demchenko et al. 2014)[xxiv]
[HPS+16]	Reference Architectures	✔	Reference of (Bakshi 2012)
[Gee13]	Reference Architectures	✔	Search keyword
[MW15]	Reference Architectures	✔	Search keyword
[MPCAF15]	Reference Architectures	✔	Reference of [PP15]
[SCG+17]	Reference Architectures	—	Search keyword
[Ama13]	Architecture Implementation	✔	Search keyword
[SKS13]	Architecture Implementation	✔	Reference of [PP15]
[MDL+13]	Architecture Implementation	✔	Reference of [PP15]
[TSA+10]	Architecture Implementation	—	Reference of [PP15]

[xxiv] The NBDRA was not yet finished as of publication time and only referenced by name. Thus, a specific search for this publication of the NIST was conducted.

Other publications in this category discuss Big Data architectures generally, but do not propose or implement an architecture.

Other results describe various different forms of architectures in the Big Data domain or other related domains such as software, network, and cloud infrastructures. They are not used for further analysis.

The relevant results according to this categorization are listed in Table A.2. Results that were not discarded during search as described above, but are not in scope for the analysis in this work are listed in Table A.3.

Table A.3: Out of scope results of the literature search on Big Data architectures, which do not contain implementations or reference architectures, but were not immediately discarded during search.

Author, Year, and Title	Category	Publication	Exclusion Reason
Bakshi, 2012. Considerations for big data: Architecture and Approach	Arch. Comp. and Tech.	Article	Category
Zhang, 2012. Big Services Era: Global Trends of Cloud Computing and Big Data	Arch. Comp. and Tech.	Editorial	Category
Lenke et al., 2016. Comparative Analysis of SpatialHadoop and GeoSpark for Geospatial Big Data Analytics	Arch. Comp. and Tech.	Scientific Report	Category
Kozlov, 2010. Hadoop/Hbase Capacity Planning	Arch. Comp. and Tech.	Blog Entry	Category
George, 2009. Hbase Architecture 101 - Storage	Arch. Comp. and Tech.	Blog Entry	Category
Demchenko et al., 2014. Defining architecture components of Big Data Ecosystems	Arch. Comp. and Tech.	Paper	Category
Gadepally et al., 2016. The BigDaWG Polystore System and Architecture	Arch. Comp. and Tech.	Scientific Report	Category
Avinash et al., 2010. Cassandra — A Decentralized Structure Storage System	Arch. Comp. and Tech.	Article	Category
Membrey et al., 2013. A disk based stream oriented approach for storing big data	Arch. Comp. and Tech.	Paper	Category
Vora et al., 2011. Hadoop — Hbase for Large-Scale Data	Arch. Comp. and Tech.	Article	Category

Table A.3: Out of scope results of the literature search on Big Data architectures (continued).

Author, Year, and Title	Category	Publication	Exclusion Reason
DeCandia et al., 2007. Dynamo — Amazon's Highly Available Key-Value Store	Arch. Comp. and Tech.	Paper	Category
Zhong et al., 2013. On Mixing High-Speed Updates and In-Memory Queries	Arch. Comp. and Tech.	Paper	Category
Meier, 2013. Towards a Big Data Reference Architecture	Ref. Arch.	Master Thesis	Publication
Wei et al., 2016. OpenCluster: A Flexible Distributed Computing Framework for Astronomical Data Processing	Other	Article	Category
Yi et al., 2014. Building a network highway for big data: architecture and challenges	Other	Article	Category
Demenchenko et al., 2013. Intercloud Architecture Framework for Heterogeneous Multi-Provider Cloud based Infrastructure Services Provisioning	Other	Article	Category
Gorton & Klein, 2015. Distribution, Data, Deployment: Software Architecture Convergence in Big Data Systems	Other	Article	Category
Balac et al., 2015. NIST Volume 2 — Taxonomies	Other	Technical Report	Category
Basanta-Val et al., 2016. Architecting Time-Critical Big-Data Systems	Other	Scientific Report	Category

Table A.3: Out of scope results of the literature search on Big Data architectures (continued).

Author, Year, and Title	Category	Publication	Exclusion Reason
Ramaswamy et al., 2013. Towards a Quality-centric Big Data Architecture for Federated Sensor Services	Other	Paper	Category
Baroso, 2009. The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines	Other	Book	Category

B Supplementary GOBIA Models

This appendix includes supplementary models of the GOBIA method. It depicts the full versions of the GOBIA.DEV processes from Phase I and Phase III, split into two partial images each. The left part of GOBIA.DEV Phase I is depicted in Figure B.1, while the right part is in Figure B.2. The top part of GOBIA.DEV Phase III is depicted in Figure B.3, while its bottom part is in Figure B.4.

The full image of the semi-concrete BI architecture in the FROG AIR case from GOBIA.DEV Phase II can be found in Figure B.5.

Finally, a color-neutral variant of the GOBIA.REF technological reference architecture is depicted in Figure B.9.

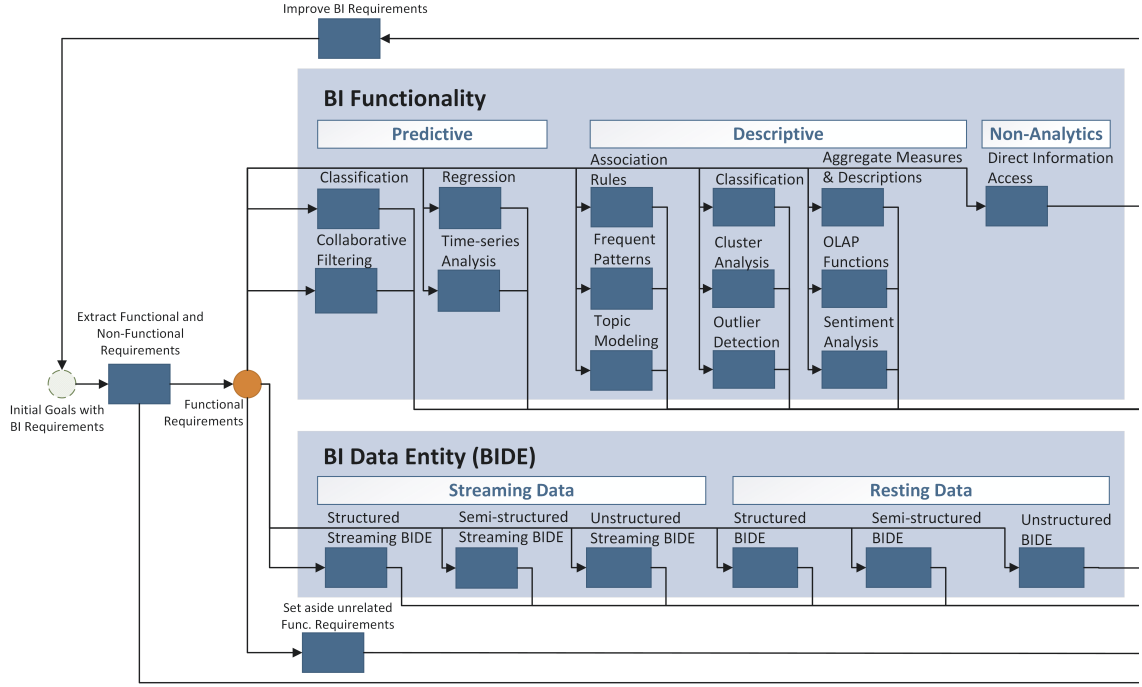

Figure B.1: Petri net process model of GOBIA.DEV Phase I, Part 1 of 2 (Left).

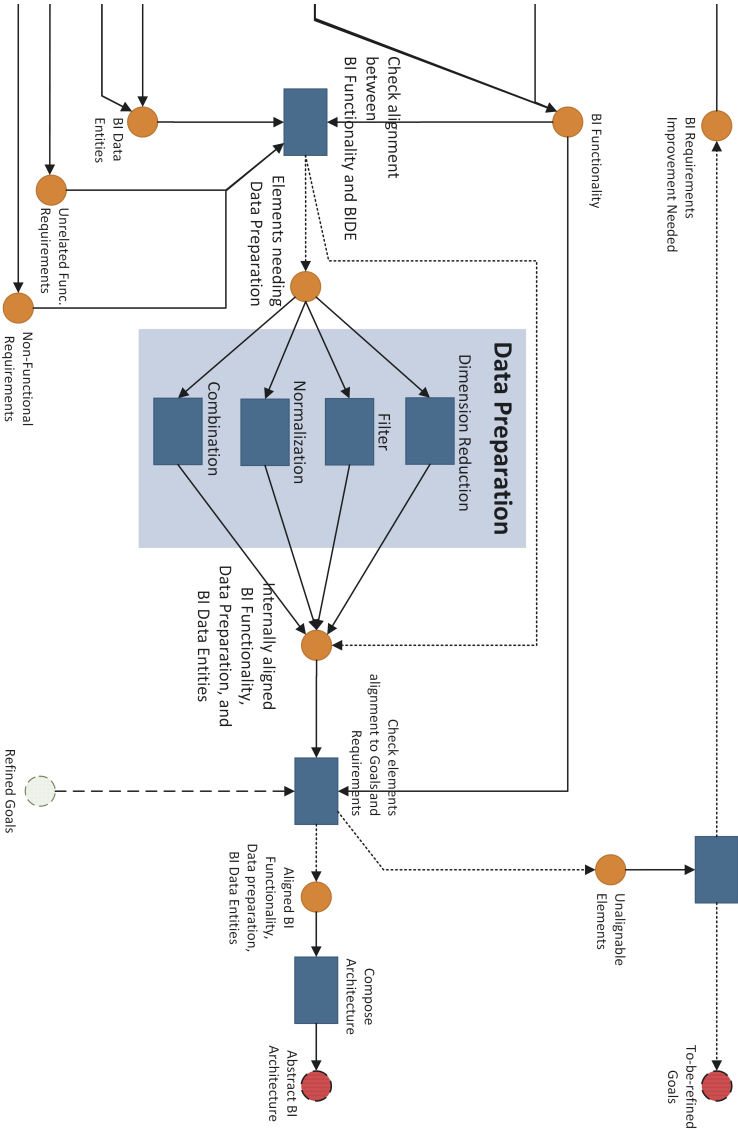

Figure B.2: Petri net process model of GOBIA.DEV Phase I, Part 2 of 2 (Right).

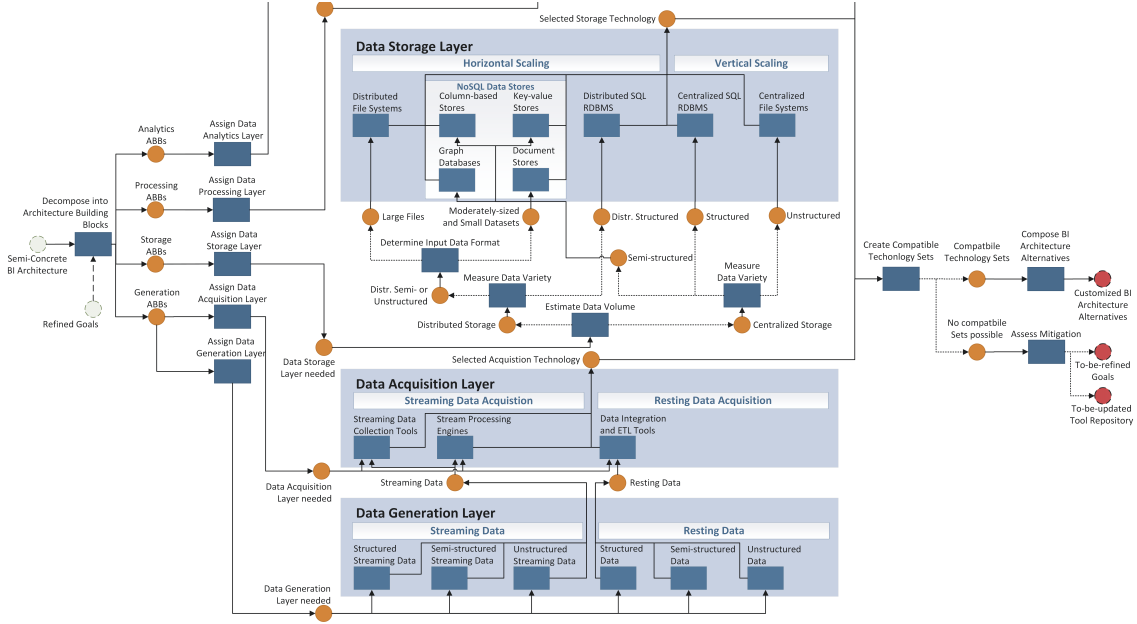

Figure B.3: Petri net process model of GOBIA.DEV Phase III, Part 1 of 2 (Bottom).

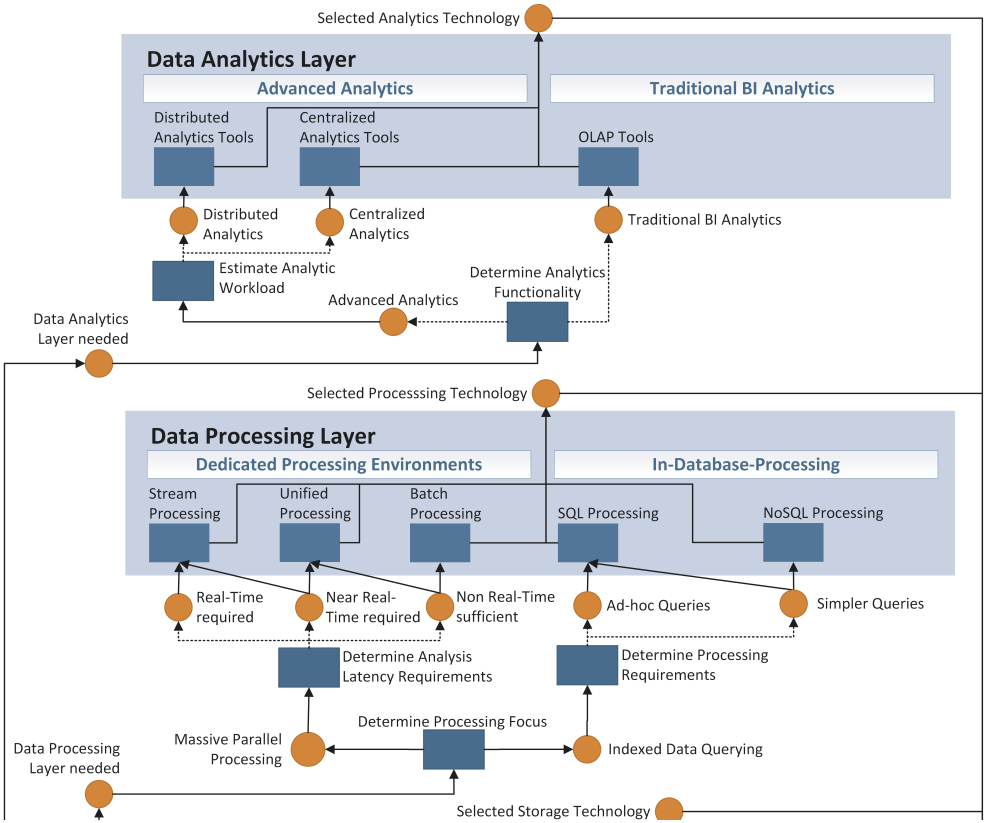

Figure B.4: Petri net process model of GOBIA.DEV Phase III, Part 2 of 2 (Top).

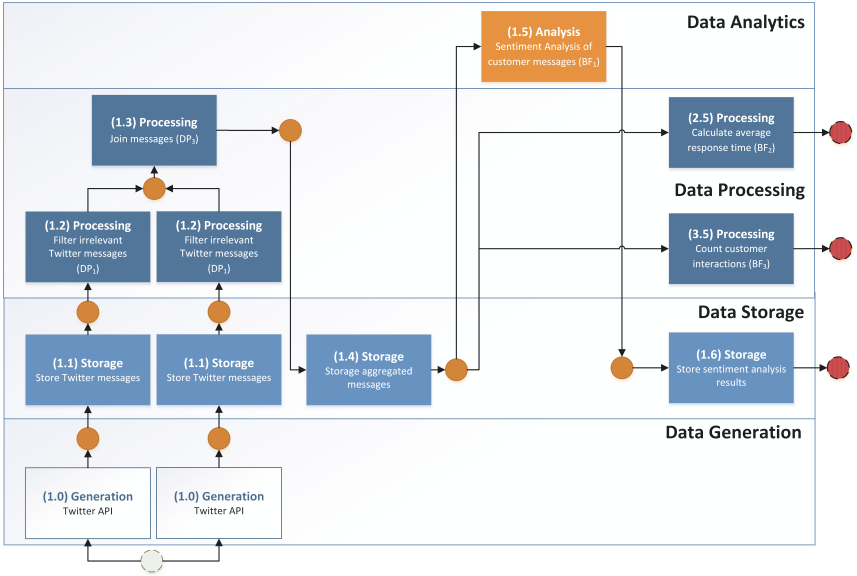

Figure B.5: FROG AIR case: Semi-concrete BI architecture as result of GOBIA.DEV Phase II depicted as Petri net. It consists of three tactical plans for three main use cases (see elements 1.5, 2.5, and 3.5), which are assigned to layers corresponding to the Architecture Building Blocks of the tactical plans they are comprised of.

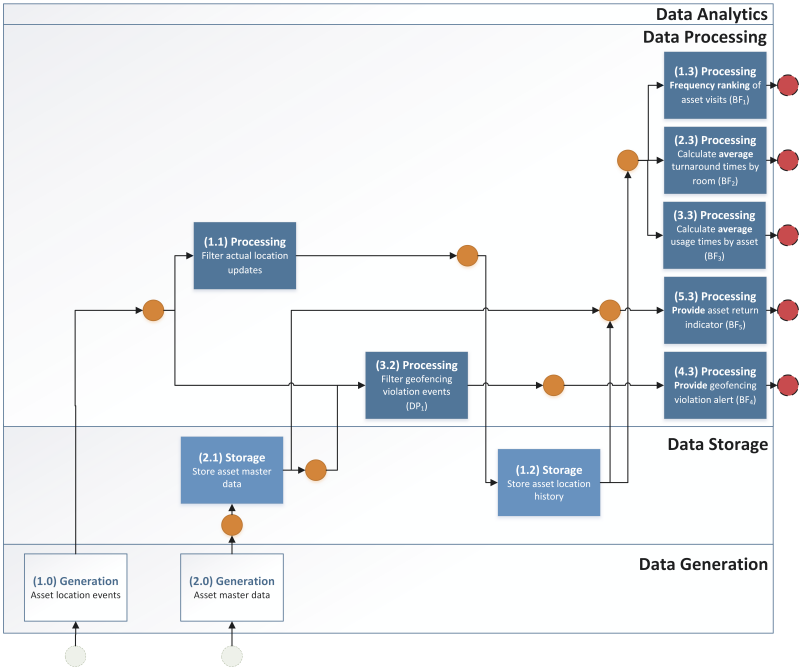

Figure B.6: HPL case: Semi-concrete BI architecture as result of GOBIA.DEV Phase II depicted as Petri net. It consists of three tactical plans for five main use cases (see elements 1.3, 2.3, 3.3, 4.3 and 5.3), which are assigned to layers corresponding to the Architecture Building Blocks of the tactical plans they are comprised of.

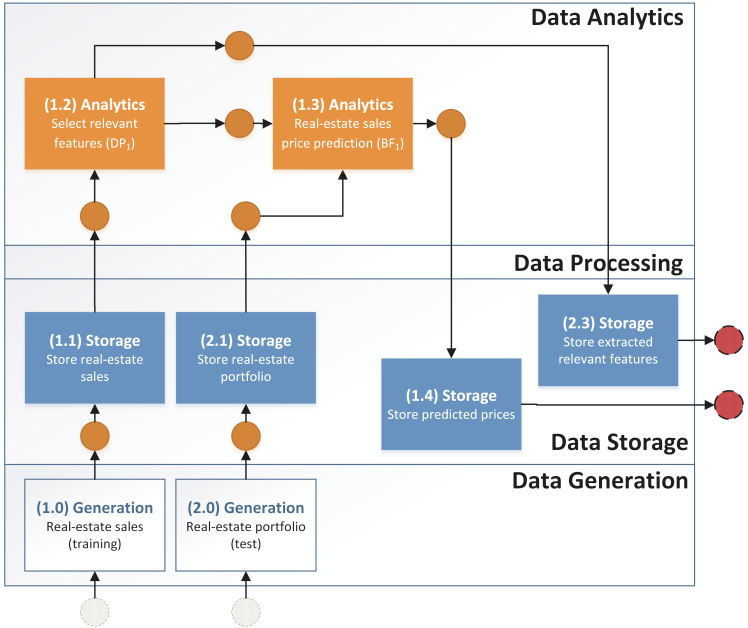

Figure B.7: KRE case: Semi-concrete BI architecture as result of GOBIA.DEV Phase II depicted as Petri net. It contains the main tactical plan for the core use case (see element 1.3), which are assigned to layers corresponding to the Architecture Building Blocks of the tactical plans they are comprised of.

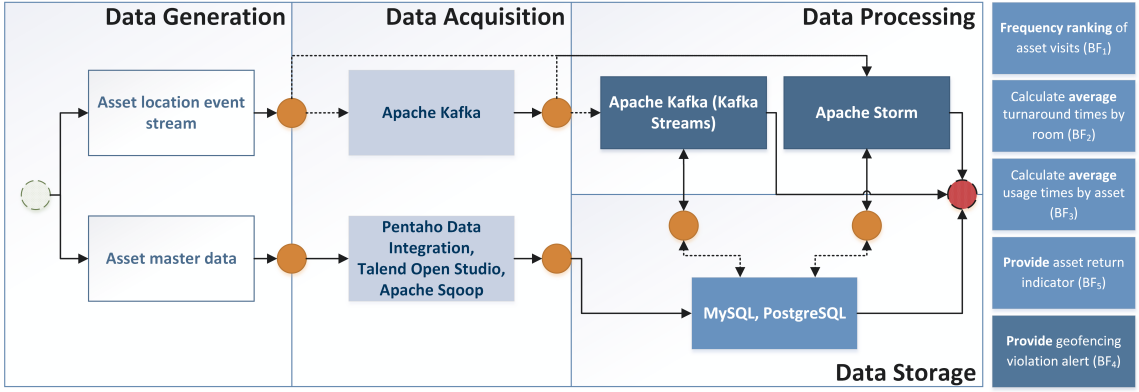

Figure B.8: GOBIA.DEV Phase III for the HPL case: Combination of all technology alternatives on all layers of Phase III. Five data uses (BF_{1-5}) are shown right to the model. Data flows are not depicted separately.

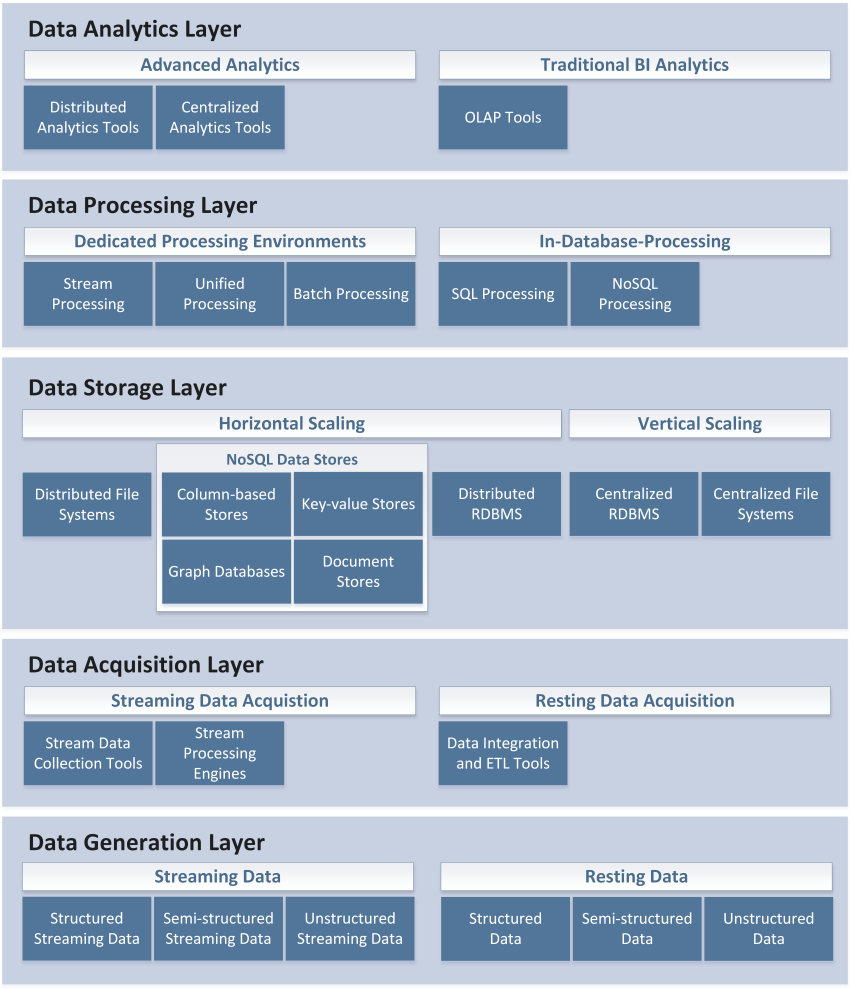

Figure B.9: Variant of GOBIA.REF Technological Reference Architecture, depicted in Figure 4.3, without additional color coding of technologies.

C GOBIA Tool Repository

First, this chapter lists all tools from the GOBIA tool repository with necessary information for selection as well as supplementary data in Section C.1. Next, the compatibility between these tools is documented in Section C.2.

C.1 Tool Lists

This section contains the tool lists of the GOBIA repository and describes their basic properties, categories, and tool-related capabilities. Besides the capabilities and categories immediately used for selection in GOBIA.DEV, supplementary data in *italics* on selected additional capabilities is provided for further illustration, e. g., for post-GOBIA evaluations in Section 4.4. Notably, the preliminary evaluations of supplementary capabilities do not represent an exhaustive (empirical) evaluation, but give an indication about additional tool capabilities. Capability support in general is indicated with a capability matrix using the following format:

- ✔ indicates that a feature is fully supported.
- (✔) indicates that a feature is partially supported.
- ✖ indicates that a feature is not supported.
- — (dash) indicates that the capability does not apply.

Sources for other attributes in the tables are mentioned in the opening paragraphs of each subsection. Additional sources are denoted for each tool or conjointly for common evaluations in the table caption.

C.1.1 Data Acquisition

The tools for data acquisition tools are listed in Table C.1. The basic technology class of data acquisition tools is derived from the descriptions in Section 4.2.2, Section 2.1.2, and Section 2.2.6 as well as vendor information. The processing class refers to the categorization of some acquisition tool with respect to their utility in Section C.1.2. Supplementary attributes include:

Data Extraction & Collection. Denotes if the tool is generally able to actively or passively ingest data. This should be ✔ for all tools in this category.

Data Flow Management. Denotes if the tool provides data flow management functionality such as queues (cf. Section 2.2.6).

Data Processing. Denotes if the tool possesses generic data processing capabilities. This includes generic manipulation of values, filtering of attributes or data points, sorting, merging of data sources, and aggregations. However, no disambiguation is made here between complete data integration and cleansing capabilities such as schema matching or data validation (cf. Section 2.1.2), because a detailed comparison of these supplementary capabilities is out of scope for this work. Full compatibility ✔ is thus warranted for this list, if aggregation, filtering, and simple manipulation capabilities could be identified for the specific tool. Partial compatibility (✔) means that various limits to processing. For example, Apache Flume can only aggregate sources in a simple manner if two workflows are connected. No other processing functionalities are available.

Data Loading. Denotes if the tool is capable to load or pass data into a sink such as another target system like a database. This should be ✔ for all tools in this category.

Scalability. Acquisition tools for vertical scaling primarily run on a single machine and perform tasks there, such as Pentaho Data Integration. Horizontally scalable tools are tuned for distributed, shared-nothing processing, such as Apache Kafka.

Graphical User Interface (GUI). This boolean attribute denotes if the tool is delivered with a GUI.

Operating Model. States if the tool works in batch mode or streaming mode. Batch tools apply transformations to a finite set of data, while streaming tools are capable of iterative or continuous executions of transformations. For streaming tools, it is outlined if their streaming model is native or based on micro-batches.

C.1.2 Data Storage

Table C.2 contains tools for data storage. Their basic categories and properties are derived from the descriptions and sources in Chapter 2, whereas additional sources are documented directly in aforementioned tables. The technology classes and basic interfaces (e. g., SQL, SQL-like or API) in query interfaces are used for technology selection in GOBIA.DEV. Here, additional capabilities are assessed and documented for further documentation and illustration of enhanced technology evaluations in post-GOBIA processes. These are noted in matrix form inside aforementioned tables. Querying capabilities are assessed through official vendor documentations. The basic approach is not to check for compliance to specific features of the SQL specification, but to assess individual feature groups such as selection or aggregation separately. Generic structure and tool selection is documented in Section 4.2. The examined capabilities are as follows:

CAP Focus. General leniency towards *CA*, *CP*, *AP*, or *CA* as described in Section 2.2.2. Specific level of consistency, availability, and partition-tolerance can be further derived from vendor documentations. Notably, trade-offs between properties might be tunable.

Access Interface. Lists approach to query data. This includes *SQL* and *SQL-like* interfaces as query languages, *APIs*, which include programmatic access such REST or REST-like APIs, as well as *File Access*, which is reserved for tools in the file system category. Here, no querying capabilities apply. Notably, programmatic access allows to access data with (imperative) functions.

Table C.1: GOBIA tool repository: Tools for Data Acquisition (1 / 2).

Tool	Pentaho Data Integration (Kettle) [205]	Talend Open Studio [374]	Apache Sqoop [99]	Apache Kafka [23]	Apache Flume [40]
Version, Date	7.1, May 2017	7.0.1, Apr 2018	1.4.7, Dec 2017	1.1.0, Mar 2018	1.8, Oct 2017
Technology Class	Data Integration and ETL Tools	Data Integration and ETL Tools	Data Integration and ETL Tools	Streaming Data Collection Tools	Streaming Data Collection Tools
Processing Class	—	—	—	—	—
Data Extraction & Collection	✔	✔	✔	✔	✔
Data Flow Management	✖	✖	✖	✔	✖
Data Processing	✔	✔	✖	(✔)[xxv]	(✔)[xxvi]
Data Loading	✔	✔	✔	✔	✔
Scalability	Vertical	Vertical	Horizontal	Horizontal	Horizontal
GUI	✔	✔	✖	✖	✖
Operating Model	Batch	Batch	Batch[xxvii]	Streaming (Native)	Streaming (Native)
Sources	[205, 129, 115]	[373, 372, 375, 376]	[99, 100]	[24, 86, 144, 361]	[361, 137, 31]

[xxv] Simple Transformations in Streams, e. g., add, filter, rename fields. [xxvi] Aggregation via workflows. [xxvii] Per Table.

Table C.1: GOBIA tool repository: Tools for Data Acquisition (2 / 2).

Tool	Apache NiFi [29]	Apache Spark (Spark Streaming) [76]	Apache Storm [13]	Apache Flink [20]	Apache Kafka (Kafka Streams) [27]
Version, Date	1.6.0, Apr 2018	2.3.0, Feb 2018	1.2.1, Feb 2018	1.4.2, Mar 2018	1.1.0, Mar 2018
Technology Class	Streaming Data Collection Tools	Stream Processing Engines	Stream Processing Engines	Stream Processing Engines	Stream Processing Engines
Processing Class	—	Unified Processing	Stream Processing	Stream Processing	Stream Processing
Data Extraction & Collection	✔	✔	✔	✔	✔
Data Flow Management	✔	✖	✖	✖	✖
Data Processing	✔	✔	✔	✔	✔
Data Loading	✔	✔	✔	✔	✔
Scalability	Horizontal	Horizontal	Horizontal	Horizontal	Horizontal
GUI	✔	✖	✖	✖	✖
Operating Model	Streaming (Native)	Streaming (Micro-Batch), Batch	Streaming (Native)	Streaming (Native)	Streaming (Native), Batch
Sources	[29, 361, 134]	[97]	[78, 77, 89, 90]	[18, 53, 56, 50]	[72]

Query: Projection. Describes if projection of data, i. e., accessing all attributes of dataset or some of them, is supported. Depending on data model, this can also be fully supported by simpler methods. For example, key-value stores can regularly access their data with simple CRUD commands. Notably, all databases and data stores examined offer this functionality.

Query: Selection. Selection capabilities denote the ability to filter rows or data sets according to certain criteria, e. g., a `WHERE` clause in SQL. When only a user-defined filter in programmatic APIs is supported, as in Apache HBase, partial support is indicated, because of added complexity of writing User-defined Functions (UDFs) instead of queries or using predefined functions.

Query: Aggregation. Aggregation capabilities are evaluated in comparison to aggregate functions described in the SQL standard [KKH08, p. 438f.]. However, as support for all possible aggregate function is not uniform even among RDBMSs, a subset of these functions is sufficient to indicate full support in the capability matrix. Namely, count, average, minimum, maximum, sum, variance, and standard deviation should be supported to be classified as fully supporting query aggregations. This selection is based on typical aggregate data descriptors and measures listed in Section 2.3.3.

Query: Joins. Join support indicates if data can be combined freely over different result sets, e. g., as with SQL `JOIN` capabilities (cf. [KKH08, pp. 9ff.]).

Query Adv.: Subqueries Besides aforementioned query capabilities, formulating subqueries (i. e., nesting queries) denotes an advanced functionality. Full support is assumed if uncorrelated and correlated subqueries can be defined.

Query Adv.: OLAP functions. OLAP functions such as grouping sets, roll-ups, and window function introduced in SQL:2003 constitute the second set of advanced querying capabilities. If support for some form of the aforementioned is given, even if not all features of the standard are implemented, full support is indicated. In the current tool selection, only MySQL 8.0 and PostgreSQL offer sophisticated OLAP function support, while other data-

bases and data stores do not offer any support for them. This also applies to MySQL 5.7, which is the basis for MySQL NDB Cluster.

Indexing. Indexes offer structures to quickly access data with low latencies. Several types of indexes can usually be defined such as indexes on arbitrary attributes, composite indexes, or full-text indexes. However, indexing supports varies greatly in NoSQL data stores and distributed RDBMSs in comparison to traditional RDBMSs, where not all types of indexes are supported or not all attributes can be easily indexed.

C.1.3 Data Processing

The tool list for the data processing layer in Table C.3 is comprised of several tools, which are primarily selected on other layers, because these provide processing *capabilities*. Here, Hadoop MapReduce remains as only original tool in this list and represents *Batch Processing*. Stream processing tool are loaded from the data acquisition tool lists, where the *Processing Class* is *Unified Processing* or *Stream Processing*. In-database processing tools are loaded from the storage tool list, where the *Technology Class* is *RDBMS* or *NoSQL data store*. This table adds the attributes *Technology Class* (cf. Section 4.2.2) and *Processing Environment*, which can refer to *In-Database-Processing* or *Dedicated Processing Environments*. The former is used when the tool originates from a database on the storage layer, whereas dedicated processing environments refer to tools from the acquisition layer, which are not used storage. This are in particular tools related to stream or unified processing, which use in-memory processing. Supplementary attributes are according to the source tables for the tool. Additionally, several supplementary attributes for stream processing are defined in Section 2.2.6, e. g., delivery guarantees or handling of out-of-order processing.

C.1.4 Data Analytics

Table C.4 lists tools for the data analytics layer. Basic properties, such as mode of operation (centralized or distributed) are listed. In addition to that, support for analytics functionalities as outlined Section 4.2 is verified using a capa-

Table C.2: GOBIA tool repository: Tools for Data Storage (1 / 4).

Tool	MongoDB [281]	Couchbase [152]	Redis [350]	Riak KV [111]
Version, Date	3.6.4, Apr 2018	5.1.0 Enterprise, Feb 2018	4.0.9, Mar 2018	2.2.3, Apr 2017
Technology Class	NoSQL data stores	NoSQL data stores	NoSQL data stores	NoSQL data stores
Technology Subclass	Document Stores	Document Stores	Key-value Stores	Key-value Stores
Primary Storage	Disk	Disk	Memory	Disk / Memory
CAP Focus	CP	AP	CA[xxviii]	AP
Access Interface	API	SQL-like (N1QL), API	API	API
Query: Projection	✔	✔	✔	✔
Query: Selection	✔	✔	(✔)[xxix]	(✔)[xxx]
Query: Aggregation	(✔)[xxxi]	(✔)[xxxii]	✖[xxxiii]	✖
Query: Joins	(✔)	(✔)	✖	✖
Adv. Query: Subqueries	✖	(✔)	✖	✖
Adv. Query: OLAP functions	✖	✖	✖	✖
Indexing	(✔)	(✔)	✖	(✔)
Sources	[282, 276, 277, 280, 279]	[148, 149, 150, 151, 147]	[345, 349, 219]	[220, 113, 110, 112, 114]

[xxviii] AP with Redis Cluster. [xxix] Querying inside values. [xxx] Querying inside values. [xxxi] COUNT, DISTINCT + manual pipeline with, e. g., Abs, Avg, Max, Min, Sum. [xxxii] SUM, AVG, COUNT, MAX, MIN. [xxxiii] Only inside values.

Table C.2: GOBIA tool repository: Tools for Data Storage (2 / 4).

Tool	**HDFS** [103]	**Apache Cassandra** [47]	**Apache HBase** [69]
Version, Date	3.1.0, Apr 2018	3.11.2, Feb 2018	1.4.3, Apr 2018
Technology Class	File Systems	NoSQL data stores	NoSQL data stores
Technology Subclass	Distributed File Systems	Column-based Stores	Column-based Stores
Primary Storage	Disk	Disk	Disk
CAP Focus	CP	AP	CP
Access Interface	File Access	SQL-like (CQL)	API
Query: Projection	–	✔	✔
Query: Selection	–	✔	(✔)[xxxiv]
Query: Aggregation	–	(✔)[xxxv]	✖
Query: Joins	–	✖	✖
Adv. Query: Subqueries	–	✖	✖
Adv. Query: OLAP functions	–	✖	✖
Indexing	–	(✔)	✖
Sources	cf. Section 2.2.3	[49, 156, 171, 48]	[94, 70, 71], cf. [CDG+08]

[xxxiv] Custom SCAN function. [xxxv] Count, Max, Min, Avg, UDFs.

Table C.2: GOBIA tool repository: Tools for Data Storage (3 / 4).

Tool	**Neo4j** [295]	**MySQL NDB Cluster** [301]	**Cockroach DB** [140]
Version, Date	3.3.5, Apr 2018	5.7.22 / NDB 7.5.10, Apr 2018	2.0.1, Apr 2018
Technology Class	NoSQL data stores	RDBMS	RDBMS
Technology Subclass	Graph Databases	Distributed RDBMS	Distributed RDBMS
Primary Storage	Disk	Memory	Disk
CAP Focus	CA	CP	CP
Access Interface	SQL-like (Cypher), API	SQL	SQL
Query: Projection	✔	✔	✔
Query: Selection	✔	✔	✔
Query: Aggregation	(✔)[xxxvi]	✔	✔
Query: Joins	(✔)[xxxvii]	✔	✔[xxxviii]
Adv. Query: Subqueries	✔[xxxix]	✔	(✔)[xl]
Adv. Query: OLAP functions	✖	✖	✖
Indexing	✔	(✔)	(✔)
Sources	[394, 292, 293, 289, 294, 290]	[298, 313, 305, 306, 307, 312]	[329, 143, 142, 141]

[xxxvi] Count, Sum, Discrete Percentile, Std. Deviation. [xxxvii] Joins are equal to relationships in a graph model; UNIONs are supported. [xxxviii] Supported, but not optimized. [xxxix] Via repeated MATCH on graph structure. [xl] Only uncorrelated.

Table C.2: GOBIA tool repository: Tools for Data Storage (4 / 4).

Tool	**MySQL** [301]	**PostgreSQL** [333]	***Centralized File System Access***
Version, Date	8.0, Apr 2018	10.3, Mar 2018	—
Technology Class	RDBMS	RDBMS	File Systems
Technology Subclass	Centralized RDBMS	Centralized RDBMS	Centralized File System
Primary Storage	Disk	Disk	Disk
CAP Focus	CA	CA	—
Access Interface	SQL	SQL	File Access
Query: Projection	✔	✔	—
Query: Selection	✔	✔	—
Query: Aggregation	✔	✔	—
Query: Joins	✔	✔	—
Adv. Query: Subqueries	✔	✔	—
Adv. Query: OLAP functions	✔	✔	—
Indexing	✔	✔	—
Sources	[308, 310, 311, 309]	[330, 331, 332]	cf. Section 4.2.2

Table C.3: GOBIA tool repository: Tools for Data Processing.

Tool	MapReduce [103]	Apache Spark[xli]	Apache Storm	Apache Flink	Apache Kafka[xlii]	MySQL	PostgreSQL	MySQL NDB Cluster	Cockroach DB	MongoDB	Couchbase	Redis	Riak KV	Apache Cassandra	Apache HBase	Neo4j
Version, Date	3.1.0, Apr 2018	*cf. Table C.1 where "Processing Class" in ("Unified Processing", "Stream Processing")*				*cf. Table C.2 where "Technology Class" in ("RDBMS", "NoSQL data store")*										
Technology Class	Batch Processing	Unified Processing	Stream Processing			SQL Processing				NoSQL Processing						
Processing Environment	Dedicated Processing Environment					In-Database-Processing										
Sources	cf. Section 2.2.4	*cf. Table C.1*				*cf. Table C.2*										
Supplementary Attributes	cf. Section 2.2.4	*cf. Section 2.2.6*				*cf. Table C.2*										

[xli] Incl. Spark Streaming. [xlii] Incl. Kafka Streams.

bility matrix. Full support is warranted if at least one dedicated method for the requested analytic function is provided. For example, if one classification algorithm is supported, the tool is denoted with ✔. More specific methods, such as association rules or collaborative filtering might be implemented manually using more generic methods such as classification or regression. For these, full support is only indicated if dedicated support is offered, e. g., a frequent itemset algorithm such as the Apriori algorithm for frequent patterns and association rules or TF-IDF for counting word frequencies in a sentiment analysis. This applies to other fields such as normalization, dimensional reduction, and outlier detection as well, where generic statistical approaches can be used individually, but only dedicated methods such as local outlier factors (LOF) for outlier detection or a principal component analysis for dimensional reduction count for full support. The assessment was conducted primarily using vendor documentation and partially verified empirically by briefly assessing the tools themselves.

C.2 Compatibility Matrix

This section details the compatibility matrix in Table C.5. The matrix is to be read from rows to columns (“left to right”). For example, a tool in a row on the left is denoted as source, and its compatibility to target the tools in the respective columns is shown in the cell. The following notation applies:

✔ The tool is fully compatible to the other tool. This means the source tool can directly access the target tool. For instance, Apache Kafka can directly make its data accessible to Apache Flink and Flink can directly consume a Kafka stream. Tools were only evaluated using officially available and endorsed connectors (e. g., officially listed as “notable” connectors for Kafka [144]). For instance, if private or organizational third parties contribute connectors, these were not considered. A tool is always considered to be compatible with itself. This does not necessarily mean that data can be transferred between separate tool instances, but that the selection process can successfully work with subsequent building blocks, which share the same tool.

Table C.4: GOBIA tool repository: Tools for Data Analytics (1 / 4).

Tool	Anaconda (R & Python) [4]	RapidMiner (Stand-alone) [340]	KNIME [232]	Weka (Stand-alone) [242]
Analytics Class	Advanced	Advanced	Advanced	Advanced
Mode of Operation	Centralized	Centralized	Centralized	Centralized
Version, Date	5.1, Feb 2018	8.1.3, Apr 2018	3.5.3, Dec 2017	3.8, Dec 2017
Classification	✔	✔	✔	✔
Regression	✔	✔	✔	✔
Collaborative Filtering	✔	✖	✖	✖
Association Rules	✔	✔	✔	✔
Frequent Patterns	✔	✔	✔	✔
Topic Modeling	✔	✔	✔	✖
Cluster Analysis	✔	✔	✔	✔
Outlier Detection	✔	✔	✔	✔
Aggregate Measures & Descriptions	✔	✔	✔	✔
OLAP Functions	✖	✖	✖	✖
Sentiment Analysis	✔	✔[xliii]	(✔)[xliv]	✔
Time-series Analysis	✔	✖	✔[xlv]	✔
Dimension Reduction	✔	✔	✔	✔
Normalization	✔	✔	✔	✔
Execution Environment	Standalone	Standalone	Standalone	Standalone
Sources	[5, 161]	[339, 272]	[231]	[241]

[xliii] With a vendor-provided extension. [xliv] Only as beta functionality. [xlv] Only as beta functionality.

Table C.4: GOBIA tool repository: Tools for Data Analytics (2 / 4).

Tool	RapidMiner (Radoop)[xlvi] [341]	Apache Mahout [75]	Spark MLlib [91]
Main Category	Advanced	Advanced	Advanced
Mode of Operation	Distributed	Distributed	Distributed
Version, Date	8.1.3, Apr 2018	0.13, 2017	2.3.0, Feb 2018
Classification	✔	✔	✔
Regression	✔	✔	✔
Collaborative Filtering	✖	✔	✔
Association Rules	✔	✔	✔
Frequent Patterns	✔	✔	✔
Topic Modeling	✔	✖	✔
Cluster Analysis	✔	✖	✔
Outlier Detection	✔	✔	✔
Aggregate Measures & Descriptions	✔	✖	(✔)[xlvii]
OLAP Functions	✖	✖	✖
Sentiment Analysis	✔	✖	✔
Time-series Analysis	✖	✖	✖
Dimension Reduction	✔	✔	✔
Normalization	✔	✔	✔
Execution Environment	Standalone	Spark, H_2O, Flink, MapReduce[xlviii]	Spark
Sources	[339, 272]	[36, 93, 28, 85, 16, 191, 32]	[92]

[xlvi] Vendor-provided extension — all RapidMiner (Standalone) notes apply as well. [xlvii] Using Spark. [xlviii] Deprecated.

Table C.4: GOBIA tool repository: Tools for Data Analytics (3 / 4).

Tool	**FlinkML** [55]	**Apache Samoa** [10]	**Apache MADlib** [33]
Main Category	Advanced	Advanced	Advanced
Mode of Operation	Distributed	Distributed	Distributed
Version, Date	1.4, Dec 2017	0.4.0-incubating, Jun 2017	1.13, Dec 2017
Classification	✔	✔	✔
Regression	✔	✔	✔
Collaborative Filtering	✖	✖	✖
Association Rules	✖	✔	✔
Frequent Patterns	✖	✔	✔
Topic Modeling	✖	✖	✔
Cluster Analysis	✖	✖	✔
Outlier Detection	✔	✖	✖
Aggregate Measures & Descriptions	✔	✖	✔
OLAP Functions	✖	✖	✖
Sentiment Analysis	✖	✖	✖
Time-series Analysis	✖	✖	✔
Dimension Reduction	✖	✖	✔
Normalization	✔	✖	✖
Execution Environment	Flink	Storm, S4, Samza, Flink	PostgreSQL, Greenplum, Apache HAWQ
Sources	[54, 57, 51]	[MSdB14, 11]	[34]

Table C.4: GOBIA tool repository: Tools for Data Analytics (4 / 4).

Tool	H2O ML (Hadoop / Spark) [195]	Pentaho Mondrian [206]	Microsoft SSAS [271]
Main Category	Advanced	Traditional BI	Traditional BI
Mode of Operation	Distributed	Standalone	Standalone
Version, Date	3.18, Feb 2018	3.14, May 2017	2017 CU6, Apr 2018
Classification	✔	✖	✔
Regression	✔	✖	✔
Collaborative Filtering	✖	✖	✖
Association Rules	✖	✖	✔
Frequent Patterns	✖	✖	✔
Topic Modeling	✖	✖	✖
Cluster Analysis	✔	✖	✔
Outlier Detection	✖	✖	✖
Aggregate Measures & Descriptions	✔	✔	✔
OLAP Functions	✖	✔	✔
Sentiment Analysis	✔	✖	✖
Time-series Analysis	✖	✖	✔
Dimension Reduction	✔	✖	✖
Normalization	✖	✖	✖
Execution Environment	Standalone, YARN, Spark	Standalone	Standalone
Sources	[194, 198, 196]	[206, 323]	[271, 266, 267, 396, 269, 270]

(✔) The tool is partially compatible to the other tool. This means that having the two tools work together requires additional efforts such as installing third-party plug-ins. Partial compatibility can also mean that a connector or feature is still in beta (e. g., as in CockroachDB [139]) or that other limitations apply, which have to be evaluated on a case-by-case basis, e. g., if loading has file type restrictions (e. g., only tabular models for most DBMSs in Microsoft SSAS [270]).

✖ The tools are not compatible to each other. This means that data cannot travel from the source to the target tools or it involves extensive efforts to do so such as writing a custom application or complex custom scripts, which go beyond data acquisition. Alternatively, another tool has to be used in between such as a regular local file system. The latter, however, can represent an acceptable trade-off after an extended evaluation.

Consideration of tool libraries and extensions. Tool libraries and extensions, such as *Kafka Streaming*, *Spark MLlib*, *FlinkML*, or RapidMiner Radoop are to be considered separately. Kafka Streaming, MLlib, and FlinkML are marked as "compatible" only towards to their respective platform. Thus, to test actual compatibility for these, the compatibility for their platform has to be checked. For example, in- and output compatibility of Kakfa Streams can be checked by examining the data for Apache Kafka. RapidMiner Radoop is an extension for RapidMiner, but runs on a separate Hive cluster and is used conjointly with RapidMiner (Standalone), which has to be considered for compatibility checks as well.

Table C.5: Compatibility Matrix between all tools in the GOBIA tool repository (1 / 3). Base sources: see tool tables; additional sources: [139, 145, 8, 101, 59, 237, 83, 338, 155, 197, 347, 128, 14, 291, 230, 233, 337, 364].

Compatibility	Pentaho Data Integration	Talend Open Studio	Apache Sqoop	Apache Kafka	Apache Flume	Apache NiFi	Apache Spark	Apache Storm	Apache Flink	Apache Kafka (Kafka Streams)	MongoDB	Couchbase	Redis	Riak KV
Pentaho Data Integration	✔	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖
Talend Open Studio	✖	✔	✖	✔	✖	✖	✖	✖	✖	✖	✔	✔	✖	✔
Apache Sqoop	✖	✔	✔	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖
Apache Kafka	✖	✔	✖	✔	✔	✔	✔	✔	✔	✔	(✔)	✔	(✔)	✖
Apache Flume	✖	✖	✖	✔	✔	✖	✔	✖	✖	✖	✖	✖	✖	✖
Apache NiFi	✖	✖	✖	✔	✖	✔	✖	✖	✔	✖	✔	✖	✖	✖
Apache Spark (Spark Streaming)	✖	✖	✖	(✔)	✖	✖	✔	✖	✖	✖	✖	✔	✖	✔
Apache Storm	✖	✖	✖	✔	✖	✖	✖	✔	✖	✖	✔	✖	✔	✖
Apache Flink	✖	✖	✖	✔	(✔)	✔	✖	✖	✔	✖	(✔)	✖	(✔)	✖
Apache Kafka (Kafka Streams)	✖	✖	✖	✔	✖	✖	✖	✖	✖	✔	✖	✖	✖	✖
MongoDB	✖	✔	✖	(✔)	✖	✔	✖	✖	✖	✖	✔	✖	✖	✖
Couchbase	✖	✔	✖	✔	✖	✖	✔	✖	✖	✖	✖	✔	✖	✖
Redis	✖	✖	✖	(✔)	✖	✖	✖	✔	✖	✖	✖	✖	✔	✖
Riak KV	✖	✔	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✔
HDFS	✔	✔	✔	✔	✖	✔	✔	✔	✖	✖	✖	✖	✖	✖
Apache Cassandra	✔	✔	✖	✖	✖	✔	✔	✔	✖	✖	✖	✖	✖	✖
Apache HBase	✔	✔	✖	✖	✖	✔	✖	✔	✖	✖	✖	✖	✖	✖
Neo4j	✖	✔	✖	✖	✖	✖	✔	✖	✖	✖	✖	✖	✖	✖
MySQL NDB Cluster	✔	✔	✔	✔	✖	✔	✖	(✔)	✖	✖	✖	✖	✖	✖
Cockroach DB	(✔)	(✔)	(✔)	(✔)	✖	(✔)	✖	(✔)	✖	✖	✖	✖	✖	✖

Table C.5: Compatibility Matrix between all tools in the GOBIA tool repository (1 / 3 — continued).

Compatibility	Pentaho Data Integration	Talend Open Studio	Apache Sqoop	Apache Kafka	Apache Flume	Apache NiFi	Apache Spark	Apache Storm	Apache Flink	Apache Kafka (Kafka Streams)	MongoDB	Couchbase	Redis	Riak KV
MySQL	✔	✔	✔	✔	✖	✔	✖	(✔)	✖	✖	✖	✖	✖	✖
PostgreSQL	✔	✔	✔	✔	✖	✔	✖	(✔)	✖	✖	✖	✖	✖	✖
Centralized File System Access	✔	✔	✔	(✔)	✖	✔	(✔)	✖	✖	✖	✔	✔	(✔)	(✔)
MapReduce	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖
Anaconda (R & Python)	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖
RapidMiner (Standalone)	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✔	✖	✖	✖
KNIME	✖	✖	✖	✖	✖	✖	✔	✖	✖	✖	(✔)	✖	✖	✖
Weka (Standalone)	✖	✖	✖	✖	✖	✖	(✔)	✖	✖	✖	✖	✖	✖	✖
RapidMiner (Radoop)	✖	✖	✖	✖	✖	✖	✔	✖	✖	✖	✖	✖	✖	✖
Apache Mahout	✖	✖	✖	(✔)	✖	✖	(✔)	✖	✖	✖	✖	✖	✖	✖
Apache Spark (MLlib)	✖	✖	✖	✖	✖	✖	✔	✖	✖	✖	✖	✖	✖	✖
Apache Flink (FlinkML)	✖	✖	✖	✖	✖	✖	✖	✖	✖	✔	✖	✖	✖	✖
Apache Samoa	✖	✖	✖	✖	✖	✖	✖	✖	✔	✖	✖	✖	✖	✖
Apache MADlib	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖
H2O ML (Hadoop / Spark)	✖	✖	✖	✖	✖	✖	✔	✖	✖	✖	✖	✖	✖	✖
Mondrian	✔	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖
Microsoft SSAS	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖

Table C.5: Compatibility Matrix between all tools in the GOBIA tool repository (2 / 3).

Compatibility	HDFS	Apache Cassandra	Apache HBase	Neo4j	MySQL NDB Cluster	Cockroach DB	MySQL	PostgreSQL	Centralized File System Access	MapReduce
Pentaho Data Integration	✖	✔	✔	✖	✔	(✔)	✔	✔	✔	✖
Talend Open Studio	✔	✔	✔	✔	✔	(✔)	✔	✔	✔	✖
Apache Sqoop	✔	✖	✖	✖	(✔)	(✔)	(✔)	(✔)	✖	✖
Apache Kafka	✔	(✔)	(✔)	✖	✔	(✔)	✔	✔	(✔)	✖
Apache Flume	✔	✖	✔	✖	✖	✖	✖	✖	✔	✖
Apache NiFi	✖	✔	✔	✖	✔	(✔)	✔	✔	✔	✖
Apache Spark (Spark Streaming)	✔	✔	✖	✔	✔	(✔)	✔	✔	✔	✖
Apache Storm	✔	✔	✔	✖	(✔)	(✔)	(✔)	(✔)	✔	✖
Apache Flink	✔	✔	✖	✖	(✔)	(✔)	(✔)	(✔)	✖	✖
Apache Kafka (Kafka Streams)	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖
MongoDB	✖	✖	✖	✖	✖	✖	✖	✖	✔	✖
Couchbase	✖	✖	✖	✖	✖	✖	✖	✖	✔	✖
Redis	✖	✖	✖	✖	✖	✖	✖	✖	(✔)	✖
Riak KV	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖
HDFS	✔	✖	✖	✖	✖	✖	✖	✖	✔	✔
Apache Cassandra	✖	✔	✖	✖	✖	✖	✖	✖	✔	✖
Apache HBase	✔	✖	✔	✖	✖	✖	✖	✖	✖	✖
Neo4j	✖	✖	✖	✔	✖	✖	✖	✖	✔	✖
MySQL NDB Cluster	✖	✖	✖	✖	✔	✖	✖	✖	✔	(✔)
Cockroach DB	✖	✖	✖	✖	✖	✔	✖	✖	✔	(✔)
MySQL	✖	✖	✖	✖	✖	✖	✔	✖	✔	(✔)
PostgreSQL	✖	✖	✖	✖	✖	✖	✖	✔	✔	(✔)

Table C.5: Compatibility Matrix between all tools in the GOBIA tool repository (2 / 3 — continued).

Compatibility	HDFS	Apache Cassandra	Apache HBase	Neo4j	MySQL NDB Cluster	Cockroach DB	MySQL	PostgreSQL	Centralized File System Access	MapReduce
Centralized File System Access	✔	✔	✔	✔	✔	✔	✔	✔	✔	(✔)
MapReduce	✖	✖	✖	✖	✔	(✔)	✔	✔	✔	✔
Anaconda (R & Python)	✖	✖	✖	✖	✔	(✔)	✔	✔	✔	✖
RapidMiner (Standalone)	✖	✔	✖	✖	✔	(✔)	✔	✔	✔	✖
KNIME	✖	✖	✖	✖	✔	(✔)	✔	✔	✔	✖
Weka (Standalone)	✖	✖	✖	✖	✖	✖	✖	✖	✔	✖
RapidMiner (Radoop)	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖
Apache Mahout	✔	(✔)	(✔)	✖	✖	✖	✖	✖	✔	✖
Apache Spark (MLlib)	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖
Apache Flink (FlinkML)	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖
Apache Samoa	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖
Apache MADlib	✖	✖	✖	✖	✖	✖	✖	✔	✖	✖
H2O ML (Hadoop / Spark)	✔	✖	✖	✖	✖	✖	✖	✖	✔	✖
Mondrian	✖	✖	✖	✖	✖	✖	✖	✖	✔	✖
Microsoft SSAS	✖	✖	✖	✖	✖	✖	✖	✖	✔	✖

Table C.5: Compatibility Matrix between all tools in the GOBIA tool repository (3 / 3).

Compatibility	Anaconda (R & Python)	RapidMiner (Standalone)	KNIME	Weka (Standalone)	RapidMiner (Radoop)	Apache Mahout	Apache Spark (MLlib)	Apache Flink (FlinkML)	Apache Samoa	Apache MADlib	H2O ML (Hadoop/Spark)	Mondrian	Microsoft SSAS
Pentaho Data Integration	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✔	✖
Talend Open Studio	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖
Apache Sqoop	✖	✖	✖	✖	✖	(✔)	✖	✖	✖	✖	✖	✖	✖
Apache Kafka	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖
Apache Flume	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖
Apache NiFi	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖
Apache Spark (Spark Streaming)	✖	✖	✔	✖	✔	✖	✔	✖	✔	✖	✔	✖	✖
Apache Storm	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖
Apache Flink	✖	✖	✖	✖	✖	✖	✖	✔	✖	✖	✖	✖	✖
Apache Kafka (Kafka Streams)	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖
MongoDB	✖	✔	(✔)	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖
Couchbase	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖
Redis	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖
Riak KV	✖	✖	✔	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖
HDFS	✖	✖	✖	✖	(✔)	✔	✖	✖	✖	✖	✔	✖	✖
Apache Cassandra	✖	✔	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖
Apache HBase	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖
Neo4j	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖
MySQL NDB Cluster	✔	✔	✔	✔	(✔)	✖	✖	✖	✖	✖	✔	✔	(✔)
Cockroach DB	(✔)	(✔)	(✔)	(✔)	(✔)	✖	✖	✖	✖	✖	(✔)	(✔)	(✔)
MySQL	✔	✔	✔	✔	(✔)	✖	✖	✖	✖	✖	✔	✔	(✔)
PostgreSQL	✔	✔	✔	✔	(✔)	✖	✖	✖	✖	✔	✔	✔	(✔)
Centralized File System Access	✔	✔	✔	✔	(✔)	✔	✖	✖	(✔)	✖	✔	✔	✖

Table C.5: Compatibility Matrix between all tools in the GOBIA tool repository (3 / 3 — continued).

Compatibility	Anaconda (R & Python)	RapidMiner (Standalone)	KNIME	Weka (Standalone)	RapidMiner (Radoop)	Apache Mahout	Apache Spark (MLlib)	Apache Flink (FlinkML)	Apache Samoa	Apache MADlib	H2O ML (Hadoop/Spark)	Mondrian	Microsoft SSAS
MapReduce	✖	✖	✖	✖	(✔)	✖	✖	✖	✖	✖	✖	✖	✖
Anaconda (R & Python)	✔	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖
RapidMiner (Standalone)	✖	✔	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖
KNIME	✖	✖	✔	✖	✖	✖	✖	✖	✖	✖	✔	✖	✖
Weka (Standalone)	✖	✖	✖	✔	✖	✖	✖	✖	✖	✖	✖	✖	✖
RapidMiner (Radoop)	✖	✖	✖	✖	✔	✖	✖	✖	✖	✖	✖	✖	✖
Apache Mahout	✖	✖	✖	✖	✖	✔	✖	✖	✖	✖	✖	✖	✖
Apache Spark (MLlib)	✖	✖	✖	✖	✖	✖	✔	✖	✖	✖	✖	✖	✖
Apache Flink (FlinkML)	✖	✖	✖	✖	✖	✖	✖	✔	✖	✖	✖	✖	✖
Apache Samoa	✖	✖	✖	✖	✖	✖	✖	✖	✔	✖	✖	✖	✖
Apache MADlib	✖	✖	✖	✖	✖	✖	✖	✖	✖	✔	✖	✖	✖
H2O ML (Hadoop / Spark)	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✔	✖	✖
Mondrian	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✔	✖
Microsoft SSAS	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✔

The Goal-oriented Business Intelligence Architectures Method

David Fekete

An abundance of new analytics technologies emerged since the advent of the Big Data trend and enabled new Business Intelligence (BI) use cases, which were not possible with traditional technologies. While Data Warehouses were the primary choice for traditional BI architectures, the technology selection and architectural patterns when constructing a customized BI architecture nowadays are less clear due the many new technological possibilities. To address this situation, design science research was conducted to outline key determinants for technology selection in BI architectures and examine architectural usage patterns including Big Data technologies. Based on this, the Goal-oriented Business Intelligence Architectures (GOBIA) method was constructed and evaluated. The GOBIA method includes a BI reference architecture, which covers both traditional and novel as well as functional and technological components. Using this, a development process supports the construction of a customized BI architecture by guiding the technology selection for a specific use case and its goals.

38,50 €
ISBN 978-3-8405-0226-2