CHAPTER

4 DOMAIN-DRIVEN DESIGN OF BIG DATA SYSTEMS BASED ON A REFERENCE ARCHITECTURE

Cigdem Avci Salma, Bedir Tekinerdogan, Ioannis N. Athanasiadis
Wageningen University, Information Technology, Wageningen, The Netherlands

4.1 INTRODUCTION

The idea of creating business value from data has always been an important concern. Many businesses have searched for ways to extract information from data in order to discover new insights and make smarter decisions. Together with the advancements of disruptive technologies such as Cloud Computing and Internet of Things, the ability to capture and store vast amounts of data has grown at an unprecedented rate which soon did not scale with traditional data management techniques. Yet, to cope with the rapidly increasing volume, variety and velocity [14] of the generated data, we can now adopt the available novel technical capacity and the infrastructure to aggregate and analyze big data. This situation has led to new and unforeseen opportunities for many organizations.

Big data has now indeed become a very important driver for innovation and growth for various industries such as health, administration, agriculture, and education [33,34]. These systems usually require considerable financial commitments and huge scale software and system deployments. To meet the business goals by means of the valorization of the big data, proper design is crucial. Unfortunately, developing big data systems is not straightforward and, despite high expectations, big data projects might fail due to the lack of selection of the right features and a proper design of the big data architecture.

In this chapter, we propose a domain-driven approach for designing big data systems based on feature modeling [15]. A feature model is a domain model that defines common and variant features of a domain. We report on our domain analysis of big data systems that has resulted in a feature model. This model includes common and variant features of big data systems and is used as a basis for designing big data architectures.

Besides the common and variant features, other drivers could also be employed throughout the methodology of the derivation of the big data software architectures such as requirements/use cases and nonfunctional requirements [31,32]. In many commercial organizations, there is a maturity curve that they go through, from using Hadoop as a cheap storage area for data they think might be useful, to specialist technology for a small group of data scientists and then to a corporate resource where people can locate useful data and perform ad hoc analysis on it. As the solutions mature, often they broaden the variety of data processed, and this brings in additional components. There is also a difference in the technology components required if the big data architecture is for analytics development, production

Software Architecture for Big Data and the Cloud. DOI: 10.1016/B978-0-12-805467-3.00004-1

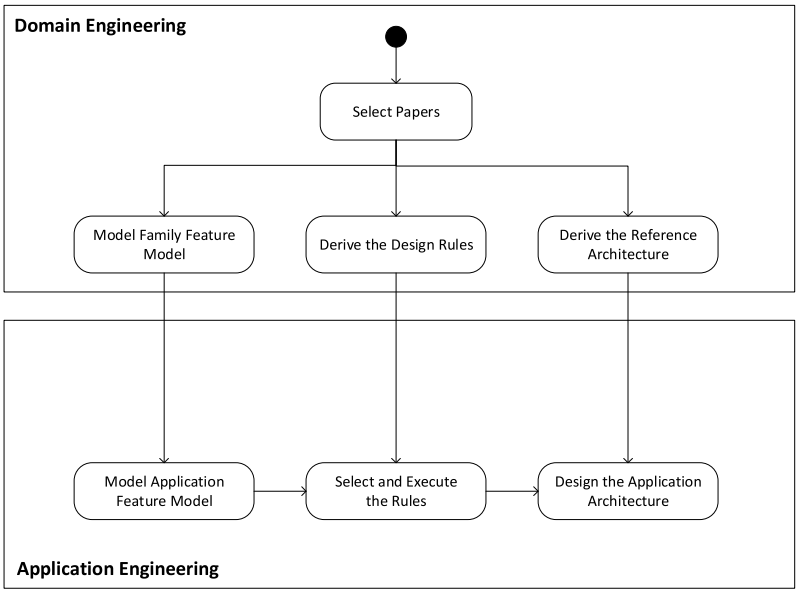

FIGURE 4.1

Adopted approach for domain-driven big data systems

use, or both. These aspects drive the roll-out of big data technology and hence of components needed for each iteration. Especially, many aspects that drive the selection of technology of the environment are driven by nonfunctional requirements. For example, the choice of processing via batch or streaming is based on the rate of data input compared with the urgency to process and consume data values. Therefore, it is important that these aspects be reflected in the model of variability. The model will be extended to drive the architecture from such business requirements as a future work.

The remainder of the chapter is organized as follows. In Section 4.2, we discuss the overall domain-driven approach that we use. Section 4.3 elaborates on the feature model for big data systems. Section 4.4 explains the approach to derive the application architectures via utilizing the feature model for big data. Section 4.5 provides the related work, and finally, Section 4.6 concludes the chapter.

4.2 DOMAIN-DRIVEN DESIGN APPROACH

The domain-driven approach for deriving big data architectures is shown in Fig. 4.1.

In essence the approach consists of two key activities, domain engineering [12] and application engineering. In the domain engineering activity, first the set of relevant papers is selected. The papers are used as input for deriving a domain model (i.e., feature model), reference architecture and the design rules. The list of papers that we have selected is shown in Table 4.1. Besides theoretical papers and white papers, we have also looked at documentation of reference architectures as defined by Big

Table 4.1 List of papers to derive the reference architecture of Big Data systems (in alphabetical order)

1	B. Geerdink, "A Reference Architecture for Big Data Solutions" [11]
2	C. Ballard et al., Information Governance Principles and Practices for a Big Data Landscape. IBM Redbooks, 2014 [3]
3	D. Chapelle, "Big Data & Analytics Reference Architecture." An Oracle White Paper (2013) [6]
4	M. Maier, A. Serebrenik, and I.T.P. Vanderfeesten, "Towards a Big Data Reference Architecture." (2013) [16]
5	NIST Big Data PWG, Draft NIST Big Data Interoperability Framework: Volume 6, Reference Architecture (2014) [17]
6	N. Marz, and J. Warren, "Big Data: Principles and best practices of scalable realtime data systems." Manning Publications Co. (2015) [18]
7	Oracle, Information Management and Big Data A Reference Architecture, An Oracle White Paper, February (2013) [20]
8	P. Pääkkönen, and D. Pakkala, "Reference Architecture and Classification of Technologies, Products and Services for Big Data Systems." Big Data Research (2015) [21]
9	S. Soares, "Big Data Governance." Information Asset, LLC (2012) [23]

Data System Infrastructure vendors. In order to assure sufficient coverage, papers were searched via and selected from three different electronic libraries which are IEEE Xplore, Google Scholar and ScienceDirect. The reference architecture in each paper among the outcome paper set is analyzed in terms of the feature set that it covers, in order to ensure that the maximum known feature coverage is reached without repetition. For feature modeling, we selected each paper of Table 4.1 and extracted the features and constraints which we used to build up the model. The feature model was evaluated after each paper by the authors.

The approach further follows the guidelines as defined by domain analysis process, which is defined as a systematic process for analyzing and modeling a domain. Domain analysis consists of Domain Scoping and Domain Modeling [13,7]. Domain scoping includes the definition of the domain of interest, the stakeholders, and their goals. In our case the domain is the domain of big data system architectures.

Based on the selected papers, the domain modeling is started, which results in a domain model. The domain model is typically derived using a commonality and variability analysis for the concepts of the selected papers. Among the domain modeling approaches, feature modeling is extensively used [13,28]. Features, their relationships, and dependencies are described in a model as a feature diagram or tables. A feature diagram is constructed as a tree, where the root symbolizes a concept (e.g., a software system), and its descendent nodes are the features. The commonalities and variations of the system properties that are aligned with the stakeholders' concerns are identified and classified as the features. The types of parent–child relationship for a feature tree are: mandatory, optional, or (one or more subfeatures have to be chosen), alternative (xor). To limit the configuration space, the feature constraints can be put in the following form (and in other ways): A excludes B.

In parallel or after the domain modeling process, the reference architecture is defined that presents the generic architecture for the various big data systems. According to Angelov et al. [1], "*a software reference architecture is a generic architecture for a class of information systems that is used as a foundation for the design of concrete architectures from this class*." Software Engineering Institute (SEI) defines the reference architecture as "*a reference model mapped onto software elements that implements the functionality defined in the reference model*" [35]; and the reference model is described

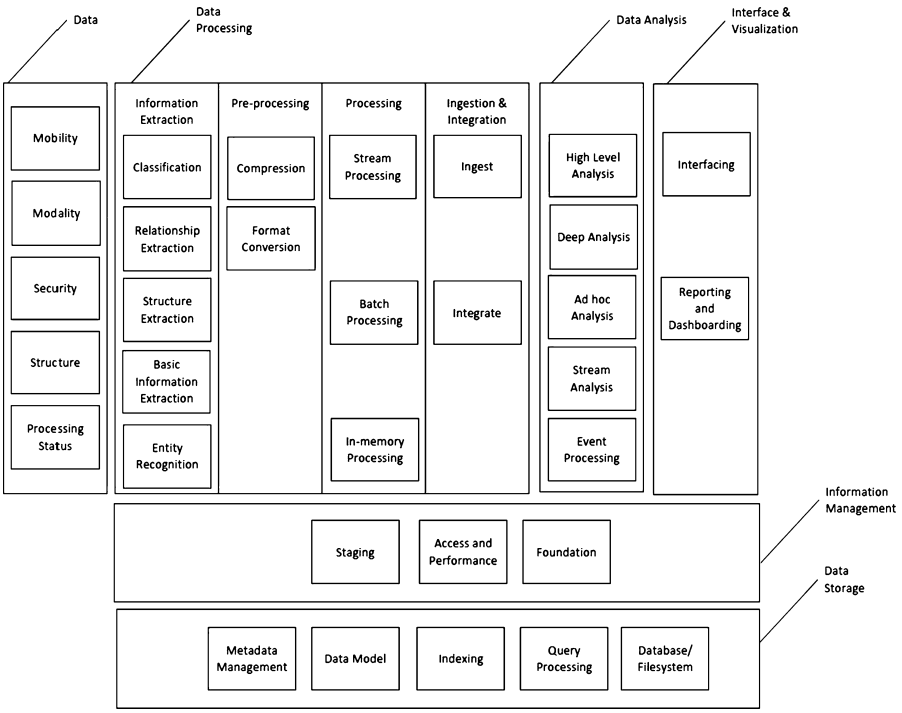

FIGURE 4.2

Big data reference architecture

as "*a division of functionality into elements together with the data flow among those elements*" [35]. A reference architecture presents the architectural best practices by various means such as standards, design patterns, and can be employed by software architects as a base from the beginning to the end of a project. The reference architecture has to be further customized to align with the requirements of the particular organization. Based on the selected papers, by means of using valid reference architectures via induction, we have defined a reference architecture for big data systems, as given in Fig. 4.2. In principle, big data systems have Data Storage, Information Management, Data Processing, Data Analysis, and Interface and Visualization components. A top level view of the reference architecture is provided in Fig. 4.1; however, each component has a more detailed coverage [22]. Especially the information management components which can be listed as catalog, security, lifecycle management, quality management, and logging are included by means of the Staging, Access and Performance and Foundation subcomponents. Often they imply a cultural change around the data use in the big data environment – which is one of the costliest parts of the big data program. We will explain these in more detail in the subsequent sections.

We can identify several other reference architectures in the literature. The reference architecture that we have derived is very similar to the other reference architectures. The reason for this is that we have considered also several other reference architectures when we defined our own reference

model. Each reference architecture typically has a different goal (e.g., explaining, describing, deriving designs). The goal of our reference architecture is to support the design of big data systems. When designing the reference architecture, we have tried to define the scope of the applications to be as broad as possible. For this purpose, we have considered the key big data concerns that are required for the majority of the big data systems. In addition, we have also considered both commonality and variability of the reference architecture. As such we aimed to achieve the generic level that is useful, in general. However, the validation of this reference architecture, like all reference architectures, is not straightforward. We have used the reference architecture for the derivation of the two different application architectures (Facebook and Twitter), and the reference architecture can also be used for other applications.

Together with the domain model and the reference architecture, we also derived the design rules from the literature. A design rule defines a design action based on a given condition. In our approach the condition is defined by a selection or deselection of a feature in the family feature diagram. A selection of a feature as such will result in the adaptation of the big data system.

In the application engineering process the family feature model, domain design rules, and reference architecture are used to derive the application architecture for a given big data system project. The family feature model covers the features of the overall big data system domain. To describe the features of a particular big data project, we derive the *application feature model* from the family feature model. The application architecture is derived using the application feature model and the reference architecture. For this the domain design rules are executed based on the selected features. Hence, a selection of different application feature models will trigger different domain design rules which will lead to a different application architecture. We define the family feature model as well as the application engineering process in more detail in the following sections.

4.3 RELATED WORK

In the literature we can identify different approaches for defining reference architectures. Galster and Avgeriou [9] propose a methodology to define empirically-grounded reference architectures. The approach consists of the following steps: decision on the type of reference architecture, selection of design strategy, empirical acquisition of data, construction of a reference architecture, enabling the reference architecture variability, and finally evaluation of the reference architecture. Our approach is at an earlier stage than that of [9]. We do not select a reference architecture from the literature but rather first define the reference architecture that is then used to derive the application architectures.

Architecture description languages (ADLs) have been proposed to model architectures. For a long time there has been little consensus on the key characteristics of an ADL. Different types of ADL have also been introduced. Some ADLs have been defined to model a particular application domain, others are more general-purpose. Also the formal precision of the ADLs differs; some have a clear formal foundation while others are less formal. Several researchers have attempted to provide clear guidelines for characterizing and distinguishing ADLs, by providing comparison and evaluation frameworks. Medvidovic and Taylor [19] have proposed a definition and a classification framework for ADL, which states that an ADL must explicitly model components, connectors, and their configurations. Furthermore, they state that tool support for architecture-based development and evolution is needed. These four elements of an ADL include other subelements to characterize and compare ADLs. The focus

in the framework is thus on architectural modeling features and tool support. In fact, we could analyze also existing ADLs based on the approach in this paper. That could be complementary to earlier evaluations of ADLs.

Architectural tactics [2] aim at identifying architectural decisions related to a quality attribute requirement and composing these into an architecture design. Defining explicit viewpoints for quality concerns can help us to model and reason about the application of architectural tactics [25]. In our approach we did not explicitly consider the notion of architectural tactics. However, we explicitly identified and described the design decisions. These can be considered similar to and complementary with architectural tactics. The advantage of the method in [2] is picking up concrete scenarios at the initial steps of design, which are also important derivers of architectural design in terms of functional/nonfunctional requirements. We would like to include the step of picking concrete scenarios in our approach as a future enhancement. On the other side in our approach, the variability of the architecture is represented in a more concrete way with the derivation of the feature model.

Architectural Perspectives [8] are a collection of activities, tactics, and guidelines to modify a set of existing views to document and analyze quality properties. Architectural perspectives as such are basically guidelines that work on multiple views together. We have primarily focused on the design of the reference architecture with respect to functional concerns. It might be interesting to look at integrating the guidelines provided by the Architectural Perspectives and the design of big data architectures. Utilizing the both approaches advantages, Domain Driven Design's concrete feature coverage and the Architectural Perspectives' quality assurance, the resulting application could have a more complete design.

Several software architecture analysis approaches have been introduced for addressing quality properties. The goal of these approaches is to assess whether or not a given architecture design satisfies desired concerns including quality requirements. For analyzing and validating the reference architecture, we have applied two real case studies (Facebook and Twitter). We were able to derive the architecture based on the reference architectures. For further more detailed analysis, we could use the existing architecture analysis approaches. We consider this as part of our future work in which we will investigate the impact of quality concerns on big data architectures.

A "description driven" approach is used to design a big data system in [10]. The crucial elements are identified and their high-level descriptions are stored in a model which is clearly separated from its instances. The description driven approach follows the principals of pure object-oriented design. In principle, our approach can be categorized as a description-driven approach since we first describe the feature model and then try to derive the design for the particular applications.

Three principles to design big data systems are listed in [4], and are: "support a variety of analysis methods", "one size does not fit all", "make data accessible".

4.4 FEATURE MODEL OF BIG DATA SYSTEMS

The top level feature diagram of big data systems that we have derived is shown in Fig. 4.3. A more detailed description of the feature diagram has been presented in our earlier work [22]. A big data system consists of the mandatory features Data, Data Storage, Information Management, Data Analysis, Data Processing, Interface and Visualization, and the optional feature, System Orchestrator. In the following subsections we will discuss each of these features in more detail.

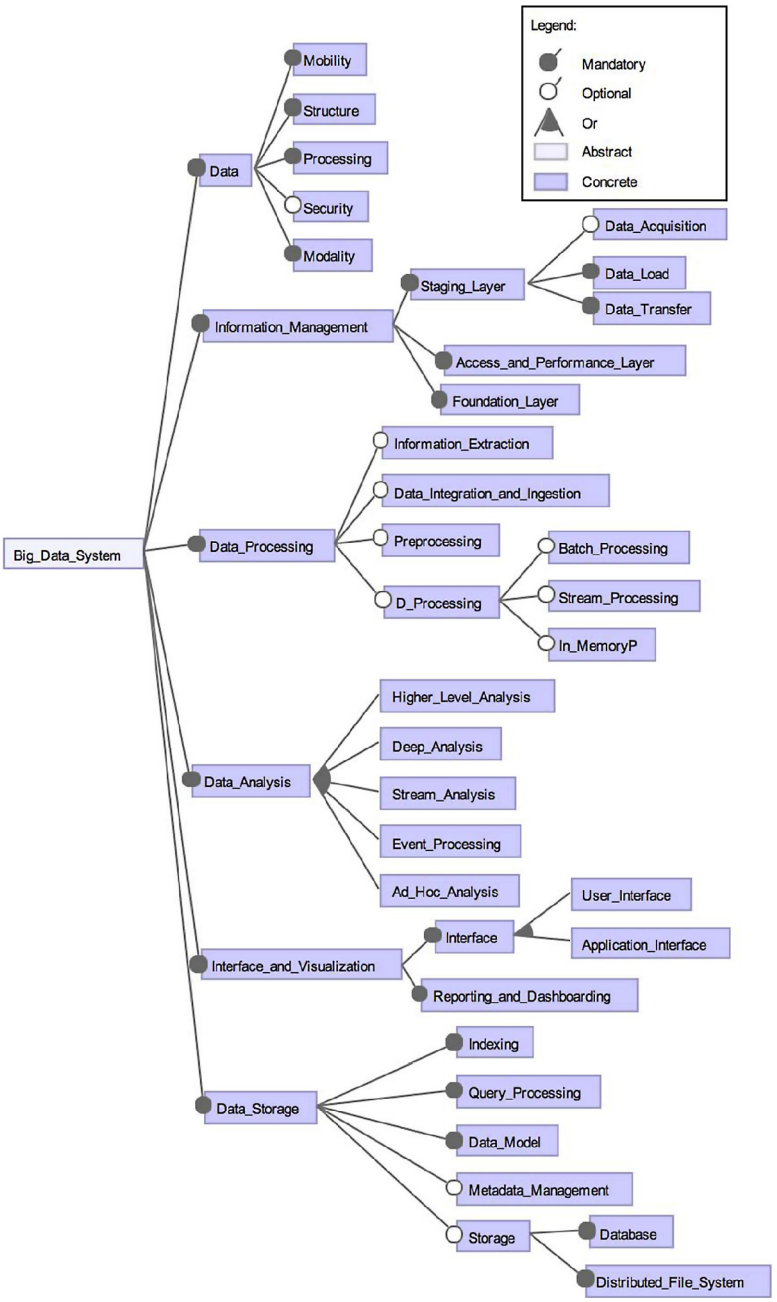

FIGURE 4.3

Top-level feature model for big data system

4.4.1 DATA

Data is the feature that defines the data types in terms of their usage, state, and representation. Hence, data in big data systems can be classified with respect to five dimensions: mobility, structure, processing, security [3], and modality. Mobility addresses the status of the data during processing and analysis activities and can be either *batch* or *streaming*. While the design of the batch processing modules should be aligned with the quality goals in terms of scalability, the design of the stream processing activities affects the performance of the system. In [18] and [6] mobility is referred as a feature of the reference architecture.

Another subfeature of the Data feature in big data systems is Structure. Depending on the source of the data, it can be of the following three structural phases: Unstructured, Semistructured, or Structured. The formation of the data processing, analysis and storage modules highly depends on the structure of the data.

The *Processing* feature defines the processing state of the data. Initially, the data in big data systems is *raw*, and can be *processed* and *analyzed*. From the data integrity perspective, different use cases can call for different level of trust [3]. Some other possible modalities of big data are textual, audio and video.

Security is not explicitly discussed in the early big data reference architectures, which glossed over this difficulty. In order to cover the early reference architectures, the security feature of the data is presented as optional. But it should be implied that nowadays it is a major part of the big data landscape, particularly in the industries where data about people and financial/intellectual property is managed by the big data solutions.

4.4.2 INFORMATION MANAGEMENT

The *Information Management* feature represents the governance of the data in terms of security, privacy, integration, and quality. It is composed of three subfeatures, and is typically implemented as layers, namely the staging, access and performance, and foundation layers. The staging layer is an abstraction of the data acquisition process. It calibrates the data receiving rate and prepares data for further processing. The access and performance layer is utilized for data access and navigation. Finally, the foundation layer isolates data in storage from the business processes so that data is ensured to be resilient to changes. (Please see [22] for more details.)

4.4.3 INTERFACE AND VISUALIZATION

The *Interface and Visualization* feature provides interaction of the big data system with the user and other applications. While reporting and dashboarding feature is used only for information presentation, the user and application interface features can provide interactive services for the users and applications. The retrieval of data via the user interfaces will require interactive response, which needs advanced optimization techniques. (Please see [22] for more details.)

4.4.4 DATA PROCESSING

Preprocessing steps, such as compression, aim to prepare data and to facilitate processing activities. Information supply chains within the big data environment that refines data from its source format into

a variety of different consumable formats for analysis and use are also covered within preprocessing activities, such as format conversion. Depending on the state of the data, processing can be classified either as stream processing (e.g., filtering, annotation) or batch processing (e.g., cleaning, combining and replication). For further processing, depending on the requirements of the system, information extraction, data integration, in-memory processing, and data ingestion activities can be employed.

Classification, entity recognition, relationship extraction, and structure extraction can be listed among the information extraction features. Data fusion, entity recognition and schema integration are the basic data integration activities. Under the category of in-memory data processing, as opposed to processing in the hard disks, high speed query processing and results caching are among possible features to be utilized. Furthermore, data ingestion which involves data obtaining and processing activities for later use is another optional data processing feature. (Please see [22] for further details.) (See also Fig. 4.4.)

4.4.5 DATA STORAGE

Data storage feature consists of query processing, indexing, distributed file system, data model, metadata management, and database subfeatures. Besides traditional data storage features, such as metadata management, relational model and relational database, a big data system can also employ features for streaming data (i.e., in-stream query processing) and for storing various data structures, with nonrelational models (i.e., NoSQL).

4.4.6 DATA ANALYSIS

Data analysis is one of the major features of a big data system. Stream analysis, high level analysis, ad hoc analysis, event processing, and deep analysis are its subfeatures that take part in the reference architectures in the selected papers (see Fig. 4.5).

4.4.7 FEATURE CONSTRAINTS

Based on the family feature diagram, we can derive many different feature configurations. However, not all feature combinations are feasible and need to be ruled out when defining the application feature model. To this end, feature constraints are used to limit the configuration space so that invalid configurations are prevented. Obviously, features which do not have a parent–child relationship are not allowed to substitute each other. Initially we can consider the constraints that are related to the mobility of the data which is either streaming or batch. For query processing on streaming data such as in sensor networks, response time gains high importance. Therefore real-time query processing is employed in such cases and as a feature constraint and real-time query processing implies streaming data. Similarly, for in-stream processing, the system does not use additional memory and does not employ time intensive storage operations, and in-stream processing implies streaming modality of the data. Besides, data acquisition feature is required to capture and analyze streaming data.

OLAP Cube is a multidimensional dataset, and OLAP data is generally stored in relational data stores or ad-hoc data management systems. While OLAP Cube implies batch processing, streaming OLAP Cubes (StreamCube) are specialized for stream processing. (See Table 4.2.)

FIGURE 4.4

Feature diagram for data processing

Data Fusion is defined as integrating and synthesizing data from multiple sources. For this feature model, we categorized the data sources depending on their modality and restricted Data Fusion to imply at least two different modalities.

Security feature of the big data systems is not discussed in detail for most of the application architectures in the known literature. In case the security feature is included in the application features, data security and privacy feature in the information management component becomes vital.

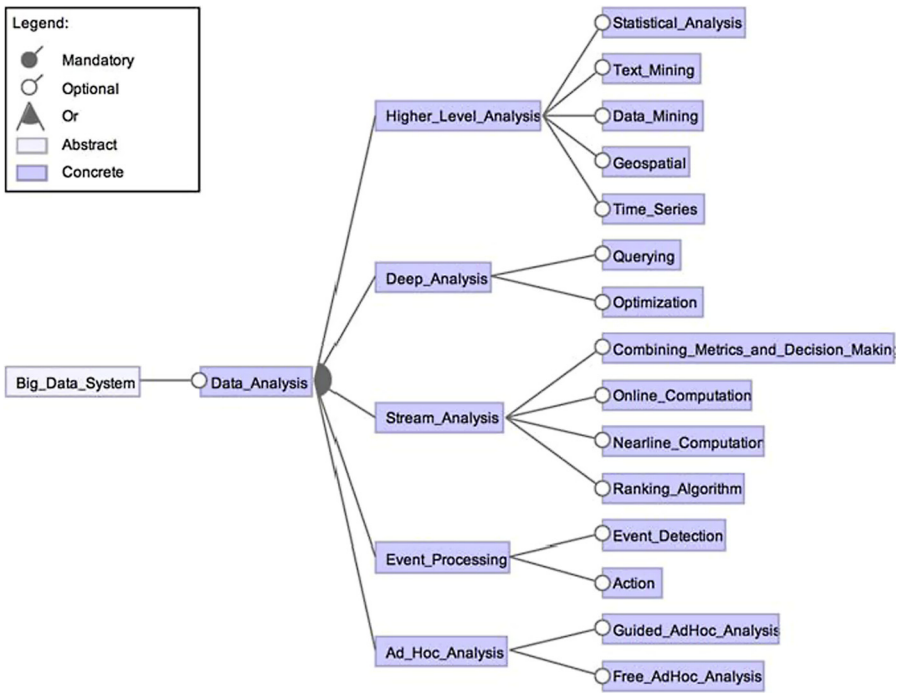

FIGURE 4.5

Feature diagram for data analysis

Table 4.2 Feature constraints

Real Time Query Processing • Streaming
In-Stream Processing • Streaming
Cube Generation • Batch
Database Snapshots • Batch
Relational Database • Relational Modal
NoSQL • Unstructured
Security • Data Security and Privacy
In-Memory File System • In-Memory processing
In-Memory Database • In-Memory Processing
Streaming • Data Acquisition
Parallel Distributed File System • Metadata Management
Object Storage File System • Data Security and Privacy
Document Data Model • NoSQL
Graph Data Model • NoSQL
Key-Value Data Model • NoSQL
Multiple Types of Data • Metadata Management
Ad Hoc Analysis • Information Virtualization
Graph Data Model • Internode Communication
Data Integration • Multiple Sources • Format Conversion

Another processing technique is in-memory processing which benefits from data stored in RAM as it is much quicker to access. In-memory databases and file systems are key features that support in-memory processing.

Schema Integration is a concept related with the distributed, heterogeneous databases which creates a global schema or access to local schemas and, together with the Parallel Distributed File Systems, requires high quality metadata. Therefore metadata management feature has to be employed when multiple data types are covered in the application architecture.

NoSQL is the category of the databases which are nonrelational. The data models other than relational data model that are listed in the feature model, namely document data model, graph data model, and key-value data model, imply the NoSQL databases.

Information virtualization is essential for ad hoc analysis because sandboxing, which is a basic virtualization technique, can support the analyst by means of isolating the data so that the data store will not be affected from a failure caused by the ad hoc analysis.

Finally, the graph data model leads to a very dense data that is distributed among the nodes. The edge information will be distributed across the boundaries as well. Therefore, to gather the edge information among the nodes, inter-node communication is extensively used, and so if the architecture of the system does not support internode communication, the system has low performance for the graph problems.

4.5 DERIVING THE APPLICATION ARCHITECTURES AND EXAMPLE

In the following sections, we define the big data application architecture derivation approach using the feature model that we defined in the previous sections. Afterwards, we apply the methodology to some well-known big data applications to derive application architectures.

4.5.1 FEATURE MODELING

In the previous sections, we defined the family feature model of the big data systems, which supports the reference architecture. We also summarized the feature constraints of this family feature model.

To derive an application architecture, considering the requirements of the application, initially the necessary features and their subfeatures are selected from the family feature model. Then the application feature model is checked against the feature constraints of the family feature model, and the inconsistency is fixed by means of including or excluding the related features from the application feature model. As an example, an application which is based on streaming data requires in-stream processing and real time querying features to be included for the data acquisition. For the same application, without having batch data requirements, selecting the OLAP cube generation feature causes inconsistency.

4.5.2 DESIGN RULE MODELING

Design rules are developed using the features and the feature constraints. The rules have the structure according to a predefined design rule definition language, i.e., “if <feature> is selected then [action] <feature> on node [name]” statement. The design rule set which covers all the features and their

constraints should be formed. Among the design rules, there can also be other reference design rules which define the connection between the derived features. Examples of design rules are listed below:

DR1: If AD HOC ANALYSIS is selected then enable INFORMATION VIRTUALISATION on component INFORMATION MANAGEMENT SERVER
DR2: If GRAPH DATABASE MODEL is selected then deploy INTER-NODE COMMUNICATION on component INFORMATION MANAGEMENT
DR3: If MULTIPLE DATA TYPES is selected then load METADATA MANAGEMENT on component DATA STORAGE
DR4: If STREAMING is selected then enable DATA ACQUISITION on component INFORMATION MANAGEMENT SERVER
DR5: If REAL-TIME QUERY PROCESSING is selected then enable DATA ACQUISITION on component INFORMATION MANAGEMENT SERVER
DR6: If IN-MEMORY DATA STORAGE is selected then enable IN-MEMORY DATA PROCESSING on component DATA PROCESSING SERVER
DR7: If OBJECT STORAGE FILE SYSTEM is selected then enable DATA SECURITY AND PRIVACY on component INFORMATION MANAGEMENT SERVER
DR8: If PARALLEL DISTRIBUTED FILE SYSTEM is selected then load METADATA MANAGEMENT on component DATA STORAGE
DR9: If DOCUMENT DATA MODEL is selected then load NoSQL on component DATA STORAGE
DR10: If GRAPH DATA MODEL is selected then load NoSQL on component DATA STORAGE
DR11: If KEY-VALUE DATA MODEL is selected then load NoSQL on component DATA STORAGE
DR12: If RELATIONAL MODEL is selected then load RELATIONAL DATABASE on component DATA STORAGE
DR13: If MULTIPLE DATA TYPES and DATA INTEGRATION is selected then load FORMAT CONVERSION on component PRE-PROCESSING

The focus of the paper is, in fact, on providing a systematic approach for deriving an application architecture using a family feature diagram and a reference architecture. We have provided the reference architecture and family feature diagram in a sufficiently detailed manner to illustrate the process. Both the reference architecture and family feature model, as well as the adopted design rules, could be further elaborated by considering additional knowledge sources. The design rules that we have adopted are used to illustrate the derivation of the application architecture for the given example applications (Facebook, Twitter). For deriving the rules, we considered the selected primary studies and looked for the reference to the features of big data systems. Very often the rules were not explicitly described, and we had to interpret and describe the design rules. As part of our future work, we will extend the design rules when considering additional primary studies.

4.5.3 ASSOCIATING DESIGN DECISIONS WITH FEATURES

So far, the application feature model has been derived and the design rule set has been developed. The next step is determining the design rule set for the application feature model. For this purpose, the selected features in the application feature model are matched with the features included in the definitions of the design rules. The matching subset of the design rules is the input for the next step.

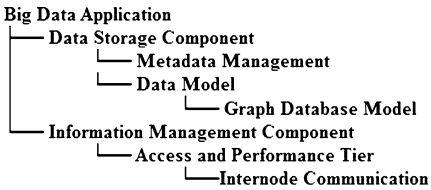

FIGURE 4.6

Big data application tree

For example, having the three rules defined in the previous section, when we select Multiple Data Types and Graph Database features for our application, the corresponding design rule set is associated with these features as follows:

Feature	Associated Design Rule
Multiple Data Types	DR3
Graph Database	DR2

4.5.4 GENERATION OF THE APPLICATION ARCHITECTURE AND THE DEPLOYMENT DIAGRAM

In this step, we have the feature model and the design rule set for the application. The aim of this step is to obtain the corresponding application architecture. In principle, this can be done manually but since the features, reference architecture, and design rules can be specified in a precise manner we could also provide automated support for this. Combining the feature model and the corresponding subset of the design rules via joining both sets on the matching features, we obtain a tree-like hierarchical structure. The structure represents the feature hierarchy and the matching design approach which is defined in the design rule set. Afterwards, these rules have to be transformed to an architecture specification. Therefore, a simple architecture description language can be defined with basic types such as device, execution, and connection. By using an ADL, the application design rules have to be converted to architectural description. Finally, the ADL instance enables the user to draw the deployment diagram.

The graphical representation of the combination of the derived features and the corresponding design rule set from the example in the previous section is shown in Fig. 4.6.

A possible ADL instance, which is obtained via transformation of our derived design rule set, can be as follows:

```
<Device name="MetadataManagement" id="4" component="DataStorage"/>
<Device name="DataModel" id="5" component="DataStorage">
<Execution id="1">GraphDatabaseModel</Execution>
<Device/>
<Device name="AccessAndPerformance" id="6" component="InformationManagement">
<Execution id="2">InternodeCommunication</Execution>
<Connection srcID="0" destID="1"/>
<Connection srcID="2" destID="1"/>
<Device/>
```

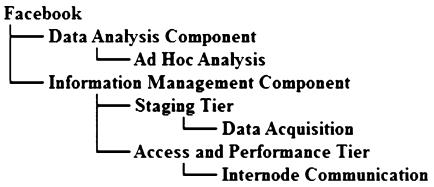

FIGURE 4.7

Facebook application tree

4.5.5 DERIVING BIG DATA ARCHITECTURES OF EXISTING SYSTEMS

In the previous section we have defined the family feature model for big data systems. The family feature model defines different possible big data systems. We can use the family feature model to characterize and define a particular big data system by selecting the corresponding features. In this section, we will illustrate this for two big data systems, namely Facebook and Twitter.

4.5.5.1 Facebook

Facebook software architecture is discussed in [29]. This architecture includes Data, Data Storage, Information Management, Data Analysis, Data Processing, and Interface, and Visualization features. Event Processing, Data Integration, Data Fusion, Entity Resolution, and Schema Integration are not defined as part of the architecture. Moreover, the architecture emphasizes the structure and mobility of data, but does not clarify its modality and security perspectives.

We cannot describe the derivation of the whole application architecture in one section; however, we are going to derive the components related to two selected features of the Facebook system. Facebook uses Scribe to collect the log data in real time which means that log data has the streaming feature. Besides, we know that Facebook system carries out ad hoc analysis on the stored data. Therefore, we select these two features from the family feature model as part of the application feature set. As the second step, we associate these two features with the related design rules from the design rule set of the family feature model as follows:

DR1: If **AD HOC ANALYSIS** is selected then enable INFORMATION VIRTUALISATION on component INFORMATION MANAGEMENT SERVER
DR2: If **STREAMING** is selected then enable DATA ACQUISITION on component INFORMATION MANAGEMENT SERVER

To generate the application architecture and the deployment diagram, we should convert the design rules to an ADL instance. (See Fig. 4.7.)

A possible ADL instance, which is obtained via transformation of our derived design rule set, can be as follows:

```
<Device name="AdHocAnalysis" id="4" component="DataAnalysis"/>
<Device name="AccessAndPerformance" id="6" component="InformationManagement">
<Execution id="2">InternodeCommunication</Execution>
<Connection srcID="0" destID="1"/>
<Connection srcID="2" destID="1"/>
<Device/>
```

```
<Device name="Staging" id="7" component="InformationManagement">
<Execution id="3">DataAcquisition</Execution>
<Device/>
```

Finally, the deployment diagram can be generated automatically by mapping the ADL instance to the DSL instance and generating the corresponding layout and connections.

We reviewed the papers related to the Facebook application architecture to check the validity of the derived application architecture [29,36,37]. In [36], the ad hoc analysis component is described as follows: "A badly written ad hoc job can hog the resources in the cluster, thereby starving the production jobs and in the absence of sophisticated sandboxing techniques, the separation of the clusters for ad hoc and production jobs has become the practical choice for us in order to avoid such scenarios." The data acquisition (gathering) component is mentioned in [37]: "The first set of applications requires realtime concurrent, but sequential, read access to a very large stream of realtime data being stored in HDFS. An example system generating and storing such data is Scribe, an open source distributed log aggregation service created by and used extensively at Facebook. Previously, data generated by Scribe was stored in expensive and hard to manage NFS servers. Two main applications that fall into this category are Realtime Analytics and MySQL backups." Anjos et al. [38] discuss the internode communication: "Distributing the data introduces other complications such as latency when communication is needed between nodes within a cluster or even between different clusters. Clever partitioning of the graph can minimize these problems. Instead of randomly distributing vertices of the graph across servers an algorithm could make sure that we minimize the internode communication and thus the latency. A typical query in a social network like Facebook is to fetch information from all your friends (neighboring vertices). The latency of this query may be significantly lowered by localizing these vertices to the same server. Facebook utilizes this by first calculating a good partitioning using Giraph and then distributing the information in their relational databases according to the suggested partitioning."

4.5.5.2 Twitter

The features of the Twitter data are listed as streaming and unstructured [27]. Data security and modality are not included in the architectural design. Especially real-time query processing and metadata management are important features of the system, while its data integration capability is limited. The architecture also employs in-memory processing. Ingestion is covered in the architectural descriptions. We use the real-time query processing [30] and in-memory data storage features of the family feature model for deriving the related part of the application architecture of Twitter. The related design rules from the design rule set of the family feature model are as follows:

DR1: If **REAL-TIME QUERY PROCESSING** is selected then enable DATA ACQUISITION on component INFORMATION MANAGEMENT SERVER
DR2: If **IN-MEMORY DATA STORAGE** is selected then enable IN-MEMORY DATA PROCESSING on component DATA PROCESSING SERVER

Afterwards, we convert the design rules to an ADL instance to generate the application architecture and the deployment diagram. (See Fig. 4.8.)

A possible ADL instance, which is obtained via transformation of our derived design rule set, can be as follows:

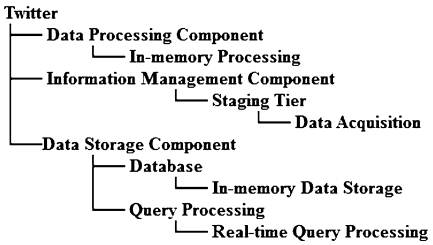

FIGURE 4.8

Twitter application tree

```
<Device name="InMemoryProcessing" id="4" component="DataProcessing"/>
<Device name="Database" id="6" component="DataStorage">
<Execution id="2">InMemoryDataStorage</Execution>
<Device/>
<Device name="Staging" id="7" component="InformationManagement">
<Execution id="3">DataAcquisition</Execution>
<Device/>
```

In [38], Twitter's in-memory processing component, which is also included in the derived application architecture, is discussed as follows: "An interesting design decision we made early in the Wtf project [5] was to assume in-memory processing on a single server. At first, this may seem like an odd choice, running counter to the prevailing wisdom of "scaling out" on cheap, commodity clusters instead of "scaling up" with more cores and more memory. This decision was driven by two rationales: first, because the alternative (a partitioned, distributed graph processing engine) is significantly more complex and difficult to build, and, second, because we could! We elaborate on these two arguments below. In 2010, our calculations showed that storing the entire Twitter graph in memory on a single machine was feasible." The data acquisition is mentioned in [24]: "The input stream of a Storm cluster is handled by a component called a spout. The spout passes the data to a component called a bolt, which transforms it in some way. A bolt either persists the data in some sort of storage, or passes it to some other bolt. You can imagine a Storm cluster as a chain of bolt components that each make some kind of transformation on the data exposed by the spout." Also in-memory data storage is implied in [38]: "Twitter graph is assumed to be stored in memory on one server." Finally in [26], we can observe that Twitter architecture employs real-time processing components (servers and search engines).

4.5.5.3 Energy management system

Energy Management System (EMS) defined in [38] as a monitoring tool to track the buildings' energy consumption. Real time query processing, data acquisition, in-memory data storage, multiple types of data, and data integration features are emphasized in the architectural description of the system. It uses a traditional relational DBMS to store historical data. We use the real-time query processing, relational modal, data integration, multiple types of data, and in-memory data storage features of the family feature model for deriving the related part of the application architecture of EMS. (See also Fig. 4.9.) The related design rules from the design rule set of the family feature model are as follows:

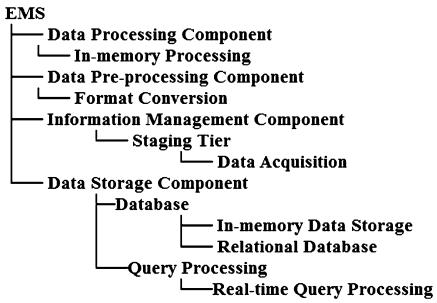

FIGURE 4.9

EMS application tree

DR1: If **REAL-TIME QUERY PROCESSING** is selected then enable DATA ACQUISITION on component INFORMATION MANAGEMENT SERVER
DR2: If **IN-MEMORY DATA STORAGE** is selected then enable IN-MEMORY DATA PROCESSING on component DATA PROCESSING SERVER
DR3: If **RELATIONAL MODEL** is selected then load RELATIONAL DATABASE on component DATA STORAGE
DR4: If **MULTIPLE DATA TYPES** and **DATA INTEGRATION** is selected then load FORMAT CONVERSION on component PRE-PROCESSING

A possible ADL instance, which is obtained via transformation of our derived design rule set, can be as follows:

```
<Device name="InMemoryProcessing" id="4" component="DataProcessing"/>
<Device name="FormatConversion" id="4" component="DataPreProcessing"/>
<Device name="Database" id="6" component="DataStorage">
<Execution id="2">InMemoryDataStorage</Execution>
<Execution id="3">RelationalDatabase</Execution>
<Device/>
<Device name="QueryProcessing" id="6" component="DataStorage">
<Execution id="5">RealTimeQueryProcessing</Execution>
<Device/>
<Device name="Staging" id="7" component="InformationManagement">
<Execution id="4">DataAcquisition</Execution>
<Device/>
```

4.6 CONCLUSION

Big data has become a very important driver for innovation and growth for various application domains. They impose different requirements on the big data system. Designing a big system as such needs to be carefully considered to realize the business goals. In this paper we have adopted a domain-driven design approach in which we provided a family feature model and a reference architecture based on a domain-analysis process. The family feature model covers the common and variant features of a

broad set of applications, while the reference architecture provides a reusable architecture for deriving concrete application architectures. We were able to derive a broad set of features that characterize multiple different big data systems. With the further development in big data research, we can enhance the feature model in the future. Similarly, we have derived the reference architecture based on a domain-analysis process. The reference architecture as such appeared to be generic and representative of the various big data systems. We have discussed the design of an application architecture that can be considered as a function of a selection of features from the family feature model that in its turn can be coupled to a design decision on the reference architecture. We have indicated that we can derive the design rules to further support this process. With further research we could derive more design rules and integrate them in the overall process for supporting the architect in deriving a feasible big data architecture. We have been able to illustrate the approach for the case of Facebook, Twitter and EMS and to derive their application architecture. We have shown that besides the specialized application architectures such as Facebook and Twitter, EMS's more generic architecture can also be successfully derived using our approach. As a lesson learned, it should be mentioned that a complete set of design rules and proper derivation of the feature model is vital to applying the approach. In our future work we plan to elaborate on the approach and provide automated support for the design of big data architectures. Besides, the model will be extended to derive the architecture from the business requirements that trigger the evolution of the software architecture of a big data system.

REFERENCES

[1] S. Angelov, P. Grefen, D. Greefhorst, A classification of software reference architectures: analyzing their success and effectiveness, in: Joint Working IEEE/IFIP Conference on Software Architecture, 2009 & European Conference on Software Architecture. WICSA/ECSA 2009, IEEE, 2009, pp. 141–150.

[2] F. Bachmann, L. Bass, M. Klein, Architectural Tactics: A Step Toward Methodical Architectural Design, Technical Report CMU/SEI-2003-TR-004, Pittsburgh, PA 2003.

[3] C. Ballard, C. Compert, T. Jesionowski, I. Milman, B. Plants, B. Rosen, H. Smith, Information Governance Principles and Practices for a Big Data Landscape, IBM Redbooks, 2014.

[4] E. Begoli, J. Horey, Design principles for effective knowledge discovery from big data, in: 2012 Joint Working IEEE/IFIP Conference on Software Architecture (WICSA) and European Conference on Software Architecture (ECSA), IEEE, 2012, pp. 215–218.

[5] P. Gupta, et al., Wtf: the who to follow service at Twitter, in: Proceedings of the 22nd International Conference on World Wide Web, International World Wide Web Conferences Steering Committee, 2013.

[6] D. Chapelle, Big Data & Analytics Reference Architecture, An Oracle White Paper, 2013.

[7] K. Czarnecki, C. Hwan, P. Kim, K.T. Kalleberg, Feature models are views on ontologies, in: 10th International Software Product Line Conference, IEEE, 2006, pp. 41–51.

[8] E. Woods, N. Rozanski, Using architectural perspectives, in: 5th Working IEEE/IFIP Conference on Software Architecture, WICSA'05, IEEE, 2005, pp. 25–35.

[9] M. Galster, P. Avgeriou, Empirically-grounded reference architectures: a proposal, in: Proceedings of the Joint ACM SIGSOFT Conference–QoSA and ACM SIGSOFT Symposium–ISARCS on Quality of Software Architectures–QoSA and Architecting Critical Systems–ISARCS, ACM, 2011, pp. 153–158.

[10] R. McClatchey, A. Branson, J. Shamdasani, Z. Kovacs, Designing traceability into big data systems, arXiv preprint arXiv:1502.01545, 2015.

[11] B. Geerdink, A reference architecture for big data solutions introducing a model to perform predictive analytics using big data technology, in: 8th International Conference for Internet Technology and Secured Transactions (ICITST), IEEE, 2013, pp. 71–76.

[12] M. Harsu, A Survey on Domain Engineering, Tampere University of Technology, 2002.

[13] K.C. Kang, S.G. Cohen, J.A. Hess, W.E. Novak, A.S. Peterson, Feature-Oriented Domain Analysis (FODA) Feasibility Study (No. CMU/SEI-90-TR-21), Carnegie-Mellon Univ./Software Engineering Inst., Pittsburgh, PA, 1990.
[14] D. Laney, 3D Data Management: Controlling Data Volume, Velocity and Variety, Meta-Group Report #949, 2001.
[15] K. Lee, K.C. Kang, J. Lee, Concepts and guidelines of feature modeling for product line software engineering, in: Software Reuse: Methods, Techniques, and Tools, Springer, Berlin, Heidelberg, 2002, pp. 62–77.
[16] M. Maier, A. Serebrenik, I.T.P. Vanderfeesten, Towards a big data reference architecture, 2013.
[17] W. May, Draft NIST Big Data Interoperability Framework: Volume 6 Reference Architecture, 2014.
[18] N. Marz, J. Warren, Big Data: Principles and Best Practices of Scalable Realtime Data Systems, Manning Publications Co., 2015.
[19] N. Medvidovic, R.N. Taylor, A classification and comparison framework for software architecture description languages, IEEE Trans. Softw. Eng. 26 (1) (2000) 70–93.
[20] Oracle, Information Management and Big Data a Reference Architecture, An Oracle White Paper, 2013.
[21] P. Pääkkönen, D. Pakkala, Reference architecture and classification of technologies, products and services for big data systems, Big Data Res. 2 (4) (2015) 166–186.
[22] C. Avci Salma, B. Tekinerdogan, I. Athanasiadis, Feature driven survey of big data systems, in: Proc. of IoTBD, April 2016.
[23] S. Soares, Big Data Governance, Information Asset, LLC, 2012.
[24] J. Leibiusky, G. Eisbruch, D. Simonassi, Getting Started with Storm, O'Reilly Media, Inc., 2012.
[25] N. Rozanski, E. Woods, Software Systems Architecture – Working With Stakeholders Using Viewpoints and Perspectives, Addison-Wesley, 2005.
[26] G. Mishne, et al., Fast data in the era of big data: Twitter's real-time related query suggestion architecture, in: Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data, ACM, 2013.
[27] B. Tekinerdogan, S. Bilir, C. Abatlevi, Integrating platform selection rules in the model driven architecture approach, in: Model Driven Architecture, Springer, Berlin, Heidelberg, 2005, pp. 159–173.
[28] B. Tekinerdogan, K. Öztürk, Feature-driven design of SaaS architectures, in: Software Engineering Frameworks for the Cloud Computing Paradigm, Springer, London, 2013, pp. 189–212.
[29] A. Thusoo, Z. Shao, S. Anthony, D. Borthakur, N. Jain, J. Sen Sarma, R. Murthy, H. Liu, Data warehousing and analytics infrastructure at Facebook, in: Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data, ACM, 2010, pp. 1013–1020.
[30] G. Mishne, J. Dalton, Z. Li, A. Sharma, J. Lin, Fast data in the era of big data: Twitter's real-time related query suggestion architecture, in: Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data, ACM, 2013, pp. 1147–1158.
[31] I. Gorton, J. Klein, Distribution, data, deployment: software architecture convergence in big data systems, IEEE Softw. 32 (3) (2015) 78–85.
[32] R. McClatchey, A. Branson, J. Shamdasani, Z. Kovacs, Designing traceability into big data systems, arXiv preprint arXiv:1502.01545, 2015.
[33] E. Begoli, J. Horey, Design principles for effective knowledge discovery from big data, in: Joint Working IEEE/IFIP Conference on Software Architecture (WICSA) and European Conference on Software Architecture (ECSA), IEEE, 2012, pp. 215–218.
[34] P. Xuan, Y. Zheng, S. Sarupria, A. Apon, SciFlow: a dataflow-driven model architecture for scientific computing using Hadoop, in: IEEE International Conference on Big Data, IEEE, 2013, pp. 36–44.
[35] Glossary, Software Engineering Institute, http://www.sei.cmu.edu/architecture/start/glossary/.
[36] A. Thusoo, et al., Data warehousing and analytics infrastructure at Facebook, in: Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data, ACM, 2010.
[37] D. Borthakur, J.S. Sarma, J. Gray, K. Muthukkaruppan, N. Spiegelberg, H. Kuang, K. Ranganathan, D. Molkov, A. Menon, S. Rash, R. Schmidt, A. Aiyer, Apache Hadoop goes realtime at Facebook, in: Proceedings of the 2011 ACM SIGMOD International Conference on Management of data, ACM, 2011, pp. 1071–1080.
[38] D. Anjos, P. Carreira, A.P. Francisco, Real-time integration of building energy data, in: 2014 IEEE International Congress on Big Data, IEEE, 2014.