Executable Architectural Models for Big Data Analytics Development and Deployment

a thesis dissertation presented
by
Cristian Camilo Castellanos
to
The Department of Systems and Computing Engineering

in partial fulfillment of the requirements
for the degree of
Doctor of Philosophy
in the subject of
Systems and Computing Engineering

Advisors: Dario Correal and Carlos A. Varela

Juries: María del Pilar Villamil and Andrés Díaz Pace

Universidad de Los Andes
Bogota, Colombia
July 2021

Executable Architectural Models for Big Data Analytics Development and Deployment

Abstract

With recent big data analytics (BDA) proliferation, enterprises collect and transform data to perform predictive analyses on a scale that a few years ago was not possible. BDA methodologies involve business, analytics, and technology domains. Each domain deals with different concerns at different abstraction levels, but current BDA development does not consider the formal integration among these domains. Hence, the deployment procedure usually implies rewriting code to be deployed on specific IT infrastructures to obtain software aligned to functional and non-functional requirements. Moreover, surveys have reported a high cost and error-prone transition between analytics development (data lab) and productive environments. This thesis explores the challenges faced by stakeholders in BDA application development and presents a domain-specific model (DSM) approach to design, validate, and generate BDA applications from an architectural perspective, bridging the gap between analytics and IT architecture domains. First, we report our survey results on BDA application deployment applied to BDA practitioners to identify current practices and challenges. Second, the ACCORDANT reference architecture with tactics and patterns catalog to facilitate BDA adoption. Next, we present ACCORDANT modeling framework, a DSM to design and deploy BDA applications via the specification of architectural inputs, functional, and deployment views. Then, we state an approach to specify and evaluate constraints using object constraints and semantic reasoning in BDA architectures conforming to ACCORDANT. Finally, we report the results of this proposal's application based on case studies and a survey that compares our approach and other according to 34 respondents.

Acknowledgments

First of all, thank God and my family, Sebastián, Lulú, my parents, and brothers who have supported and encouraged me to move forward. I should like to express my particular thanks to my advisors, Prof. Dario Correal and Prof. Carlos A. Varela, for their guidance, support, and patience both at personal and academic levels to develop this research.

My gratitude extends to my juries, Prof. María del Pilar Villamil and Prof. Andrés Díaz Pace, for their constructive comments and questions, which give me a new perspective of this work.

I would also like to thank Boris, my friend, and research partner, for long academic and personal discussions and constant support in the light of some beers. Also, thanks to the great group of Ph.D. students at Universidad de Los Andes and RPI (Rensselaer Polytechnic Institute) for passionate technical discussions and delightful moments, which helped me build and improve this work. I would also like to thank the current and former Worldwide Computing Laboratory members at RPI. They made me feel welcome during my research stay, specially Shigeru Imai and Saswata Paul.

Fulbright Colombia supported this research and the Center of Excellence and Appropriation in Big Data and Data Analytics (CAOBA), financed by the Ministry of Information Technologies and Telecommunications of the Republic of Colombia (MinTIC) through the Colombian Administrative Department of Science, Technology, and Innovation (COLCIENCIAS) within contract No. FP44842-anex046-2015.

Contents

Listing of figures

We can only see a short distance ahead, but we can see plenty there that needs to be done.

Alan Turing

1 Introduction

Big data analytics (BDA) enable enterprises to collect and transform data to carry out predictive analyses on a scale that a few years ago was not possible. To do this, paradigms such as the internet of things (IoT), No-SQL databases, open data, data streaming, machine learning (ML), map-reduce, and grid computing enable us to extract insights from raw data to add value to the business. Customer churn prediction, cross-selling, network intrusion prevention, and disease outbreaks prediction are successful industry examples of BDA applications. For these reasons, BDA investment is estimated to be a US $150.8 billion market in 2017 [52]. Useful predictions are possible by combining organizations' internal data to open data and linking the datasets in a meaningful way. BDA requires a clear business understanding, sophisticated skills, and advanced tools to extract value from information.

1.1 Big data analytics (BDA)

According to the National Institute of Standards and Technology (NIST) in [14] *big data* refers to the inability of conventional architectures to efficiently handle the characteristics of new datasets: volume (i.e., the size of the dataset); variety (i.e., data from multiple repositories, domains,

or types); velocity (i.e., rate of flow); and variability (i.e., the change in other characteristics). Hence, "Big Data consists of extensive datasets—primarily in the characteristics of volume, variety, velocity, and variability —that require a scalable architecture for efficient storage, manipulation, and analysis". This definition also refers to the quality attributes addressed by the architectures to deal with big data characteristics. These quality attributes must be thoughtfully considered to design scalable and efficient architectures supported by a high degree of storage and processing parallelism.

The NIST also defines the analytics process as the synthesis of knowledge from information. This analytics process includes collecting raw data, preparing, analyzing patterns to synthesize knowledge, and producing business value. Big data technologies expand the possible types of analytics to perform. However, they do not generate new types of analytics; instead, they enable data scientists to apply data analysis in ways that were not previously possible. BDA is defined in [14] as the ability to process and analyze information that presents big data characteristics.

The BDA applications that will be tackled in this thesis are software applications that analyze big data applying ML taking into account the data volume, variety, and velocity. These applications do not include the visualization or user interface layer, since we consider this topic belongs to the human-computer interaction (HCI) research field and falls beyond the scope of this thesis.

1.2 Challenges in BDA deployment

Analytics methodologies [24, 51] involve three knowledge domains typically: business, analytics, and technology. The business expert has to define the business goals to drive the analytics project in the business domain. In the analytics domain, these business goals are translated by data scientists into specific analytics tasks such as data cleaning, model building, and evaluation. Finally, in the technology domain, the IT (Information Technology) architect must deploy the software solution on a specific production environment regarding quality attributes (QAs) like availability, low latency, scalability, security, etc. In this context, deployment activity involves putting the analytics models into productive environments regarding software engineering discipline to operationalize BDA solutions. In this thesis, we refer to *BDA solution* or *BDA application* as a software system that stores and processes data offering analytics services. However, it does not include visualization and presentation aspects. The stakeholders (i.e., business experts, data scientists, and IT architects) deal with different concerns and vocabularies at different abstraction levels.

Recent surveys [30, 31, 81, 82] have reported that deployment challenges have not improved in recent years. Few people report their results are "always" deployed (13% in 2015). Only half of the respondents report that their analytics results are deployed "most of the time". Moreover, about a third of respondents report their models are deployed "sometimes" or less often. Forty percent of data scientists report that models are deployed within days of completing their analyses. However, for over 25% of people, model deployment takes months or years. Respondents report technical difficulties with model translation for deployment and incompatibility across multiple tools and communication problems.

On the one hand, a data scientist performs data understanding, data preparation, model building, and model evaluation iteratively until achieving an acceptable level of accuracy. The output produced by data scientists is usually code implemented in data science-driven tools focused on the functional perspective (ML techniques) developed within a controlled environment (the data lab). On the other hand, an IT architect has to translate these analytics functions and models to software components considering quality scenarios, big data tactics (i.e., [43]) and technology policies. This translation usually implies restructuring and rewriting code to obtain scalable, secure, and low-latency software components deployed on specific IT infrastructures.

Due to this, articulation and integration of end-to-end BDA solutions among these domains present some challenges. There is a lack of alignment between business domain and data analytics [85, 97, 98], and a high cost and error-prone transition between data lab and productive environment [27, 49, 100]. Although there is a growing interest in big data, successful deployments are still scarce ("Deployment Gap" phenomenon) [28].

The deployment gap can be reduced by reusing pieces of BDA software components to build new applications. However, in this line, we have the challenge of detecting and preventing "architectural mismatch" when reusing components. This architectural mismatch emerges when assumptions about their constitutive components do not match or conflict with those of other components [36]. Regarding the BDA domain, previous Architecture Description Languages (ADL) are too general or fall short of detecting and preventing architectural mismatch from different viewpoints. For instance, they do not consider the data processing model (streaming, batch), delivery semantics (at most once, at least once, exactly once), technology, and deployment mismatches. Besides, current ADLs on BDA are too generic to support tool-assisted detection of architectural mismatch.

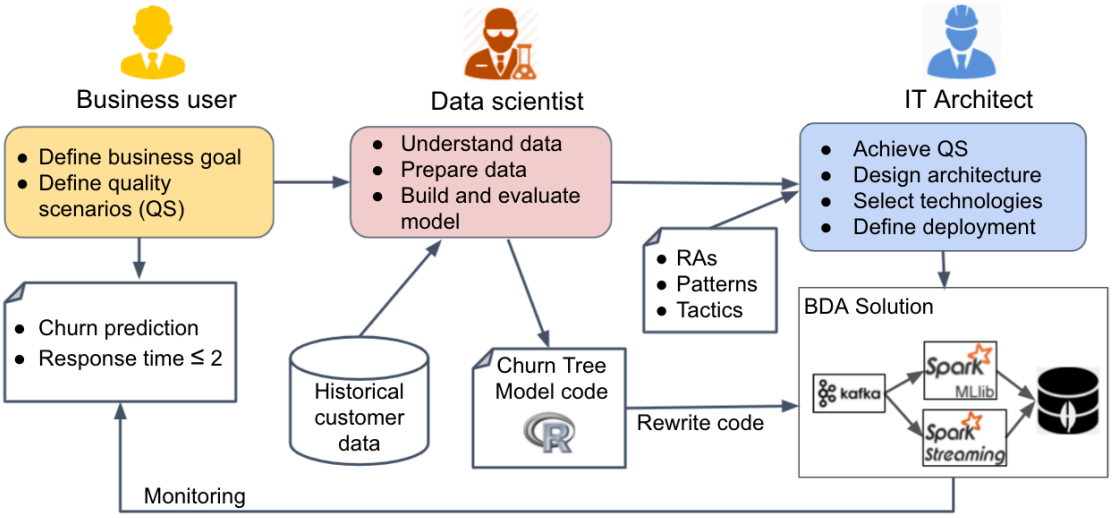

Figure 1.3.1: BDA solutions development

1.3 Motivation

We use a running example to illustrate the BDA deployment challenges presented previously to motivate our research. Fig.1.3.1 describes the BDA solutions deployment process, which covers three knowledge domains: Business, data science/analytics, and IT. For instance, a business goal in a telecom company could be: *Increase the customer retention indicators by identifying customers at high risk of churn and implementing actions to retain them*. Besides, the business user and IT architect define quality scenarios (QS) that the solution must achieve in operation. A QS is a quality-attribute-specific requirement that consists of a source of stimulus, the stimulus, environment, artifact, response, and response measure. QS guides the architecture design and evaluation. A QS aligned to the business goal could be in our example: the average response time for a request to the churn estimator system must be lower than 2 seconds under normal system operation.

In the analytics domain, the data scientist translates the business goal to analytics task, predicting the customer churn risk using a binary classification model. To do this task, the data scientist understands and prepares the historical customer data to build and evaluate a predictive model following a methodology such as CRISP-DM [24]. The data scientist tries different classification models and hyper-parameters in the data lab using a data science tool, like Scikit -Learn[1]. When a candidate model achieves an expected quality (accuracy, performance, inter-

[1]https://scikit-learn.org

pretability, fairness), this model can be deployed to a productive environment. The analytics model is traditionally source code developed in the data lab language, for instance, a decision tree model implemented in Scikit-Learn and shown in Listing 1.1. This code reads the dataset (lines 4-7), builds and trains the model (lines 8-9), and evaluates the model accuracy (lines 11-12).

Listing 1.1: Extract of Scikit-Learn code to train a decision tree model to predict churn

```
1  from sklearn import tree
2  import pandas as pd
3  ...
4  features = ['VMailMessage',..., 'Churn']
5  data = pd.read_csv('customersData.csv', names=features)
6  x = data[features[:-1]]
7  y = data['Churn']
8  clf = tree.DecisionTreeClassifier(criterion='gini', max_depth=4)
9  clf = clf.fit(x, y)
10 ...
11 scores = cross_val_score(clf, x, y, cv=5)
12 print(scores.mean(), scores.std())
```

Once the evaluated model is ready to be deployed to the production environment, IT architects design a BDA solution that implements the business goals and fulfills the QS. The architecture design is supported by architecture mechanisms such as styles, reference architectures (RA), tactics, and patterns that aim to achieve the expected QS. In addition, IT architects review and evaluate the available technologies to select the best fit for the BDA solution, considering features, licensing, and compatibility among them. For instance, the architect can decide to implement a pipe and filter architecture style implemented with Apache Spark to achieve high performance and fault-tolerance using distributed processing. A component diagram in Unified Modeling Language (UML) depicts how components are connected to form larger components or software systems. Fig. 1.3.2 illustrates a possible component diagram for the churn estimation system. Since the production deployment pursues different quality attributes that the model built in the data lab, the source code needs to be rewritten to a different language or structure. The Listing 1.2 exemplifies the code to implement the decision tree model in Apache Spark[2].

The architect also defines the deployment infrastructure to meet the goals regarding availability, security, and performance. The deployments can range from bare metal or virtual machines on-premises to containers or serverless functions in the cloud. A deployment diagram in UML models the physical deployment of artifacts on computing nodes. For example, Fig.1.3.3

[2]https://spark.apache.org/docs/latest/mllib-decision-tree.html

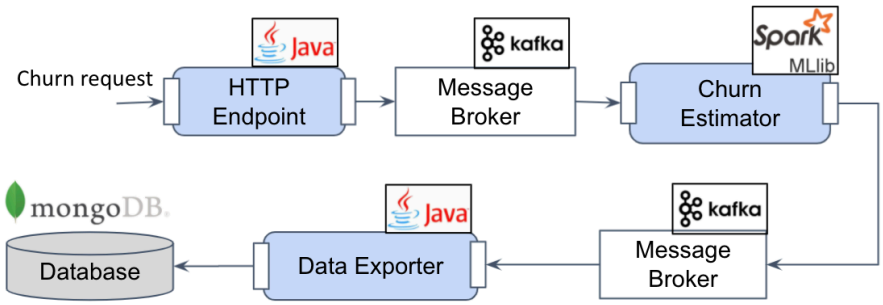

Figure 1.3.2: Component-connector diagram of the churn estimation system

presents the deployment diagram of the churn estimation system in three computing nodes, implementing three replicas for the estimation component to provide more computing resources and improve availability and performance.

Listing 1.2: Extract of Scikit-Learn code to train a decision tree model to predict churn

```
SparkConf sparkConf = new SparkConf().setAppName("ChurnDecisionTree");
JavaSparkContext jsc = new JavaSparkContext(sparkConf);
String datapath = "data/customer_data.txt";
JavaRDD<LabeledPoint> data = MLUtils.loadLibSVMFile(jsc.sc(), datapath).toJavaRDD();
...
JavaRDD<LabeledPoint>[] splits = data.randomSplit(new double[]{0.7, 0.3});
JavaRDD<LabeledPoint> trainingData = splits[0];
JavaRDD<LabeledPoint> testData = splits[1];

Map<Integer, Integer> catInfo = new HashMap<>();
String impurity = "gini";
int maxDepth = 4;
DecisionTreeModel model = DecisionTree.trainRegressor(trainingData,
  catInfo, impurity, maxDepth);
...
JavaPairRDD<Double, Double> predAndLabel =
  testData.mapToPair(p -> new Tuple2<>(model.predict(p.features()), p.label()));
double testMSE = predAndLabel.mapToDouble(pl -> {
  double diff = pl._1() - pl._2();
  return diff * diff;
}).mean();
```

Once the solution is running, the architect monitors its performance metrics to evaluate if QS are achieved. The solution evolution generates new iterations over this cycle, such as QS that are not met, new analytics model versions, or available technologies are updated. Each new iteration in the current approach involves code rewiring, testing with multiple technologies, and deployments that can cause delays and bugs injection. So, the problem of implementing multiple models versions on candidate technologies (i.e., applications) across different infrastructure

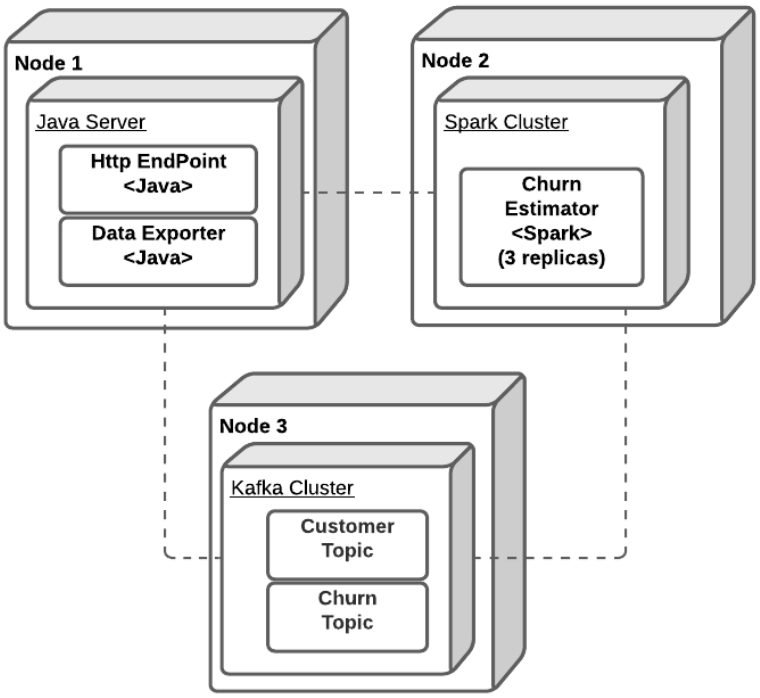

Figure 1.3.3: Deployment diagram of the churn estimation system

deployments exhibits a cubic complexity: $N = M \times A \times D \approx O(n^3)$, where M is the set of models; A is the set of applications; and D is the set of infrastructure deployments. This means that I is the set of all possible implementations, and i_t is a concrete implementation i developed in the iteration t such that $i \in I$; $i_t = (m_t \in M, a_t \in A, d_t \in D)$ as shown in Fig. 1.3.4.

As a result, each new change introduced in a new iteration comprises additional implementations, including code rewriting, testing with multiple technologies, and deployments until the expected behavior is achieved. Hence, the amount of implementations grows at an exponential rate as a function of iterations (changes) as depicted in Fig 1.3.5. This complexity in the BDA development hinders the deployment process, generates delays, and affects the quality of BDA applications.

1.4 Problem statement

The current practices and frameworks to design, evaluate, and deploy BDA applications from ML models to productive environments are cumbersome, time-consuming, and error-prone.

A software development approach to promote the increasing abstraction level and represent

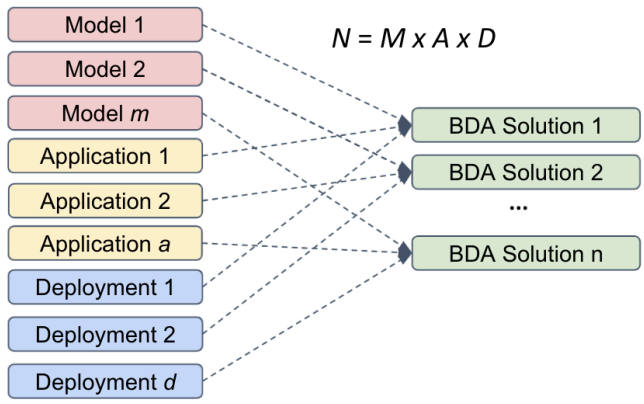

Figure 1.3.4: BDA deployment complexity

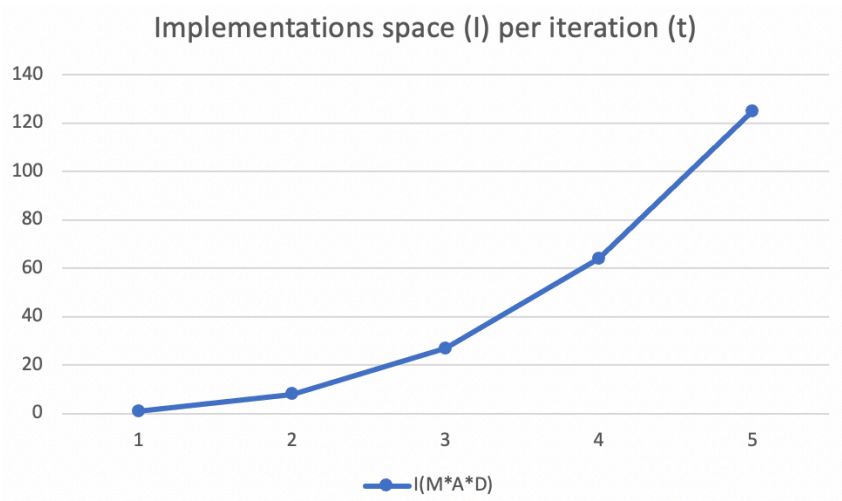

Figure 1.3.5: Growing of implementations space per iteration

the various facets is Domain-Specific Modeling (DSM). DSM aims to raise the level of abstraction beyond programming by specifying a neutral–technology domain language that directly uses concepts from the problem domain and generates final software products in a selected target technology [57]. Besides, the DSM approach enables ACCORDANT to reasoning and analyzing architectural models given formal specifications, thus allowing architects to specify constraints to detect and prevent architectural mismatches. This formalization makes explicit the architecture assumptions and allows us to validate them automatically. On the other hand, we also propose a conceptual reference architecture (RA) to guide the implementation of software architectures within a particular domain such as BDA. An RA allows the definition of domain-specific components and provides a common vocabulary to discuss different implementations. Hence, an RA helps guide software design processes and make them efficient, effective, and interoperable [5, 65].

Hence, the hypotheses of this thesis are stated in H1, H2, and H3

Hypothesis 1 (H1): *Including a DSM approach reduces the deployment time of BDA solutions.*

Hypothesis 2 (H2): *Including a DSM approach improves the architectural design and analysis of BDA applications.*

Hypothesis 3 (H3): *Implementing a DSM approach and constraints enables architects to check architectural mismatches in BDA applications.*

1.5 Research objectives

This research reviews the current challenges during the design, development, and deployment of BDA applications. To deal with challenges found in this area, we present a DSM to design data analytics architectures to bridge the gap between analytics and IT architecture domains in terms of data transformations, analytics models, software components, connectors, and deployment environments regarding predefined quality scenarios. The research objectives and their association with the hypotheses are detailed in Table 1.5.1.

1.6 Contributions

The main contribution of this thesis is a DSM approach to model, deploy, and operate BDA systems that involve data scientists and IT architect concerns. Specifically, this thesis encompasses the following contributions:

Table 1.5.1: Research objectives and related hypothesis

Research Objective	Hypothesis
RO1: Identify the current analysis, design, and deployment techniques, tools, and practices used when developing BDA applications.	*H1*, *H2*, *H3*
RO2: Review the state-of-the-art in BDA applications development, validation, and deployment to identify and compare approaches and gaps in this field.	*H1*, *H2*, *H3*.
RO3: Design a reference architecture for BDA applications to guide the implementation of concrete software architectures in this domain.	*H2*.
RO4: Provide an integrated architectural method to design, validate, and deploy BDA architectures using a Domain Specific Model (DSM) approach.	*H1*, *H2*, *H3*.
RO5: Design and develop a framework to specify independent views to decouple functional and deployment specifications on BDA architectures.	*H1*, *H2*.
RO6: Design and develop mechanisms to provision BDA technology infrastructure through deployment automation.	*H1*.
RO7: Design and develop a framework to detect architectural mismatch in BDA applications using domain constraints and semantic reasoning.	*H3*.

- A survey on BDA development and deployment practices.
- A mapping study of BDA development and deployment.
- A reference architecture for big data analytics which includes a patterns and tactics catalog.
- A domain-specific modeling approach composed by viewpoints to design and deploy BDA solutions.
- An Architecture Description Language (ADL) conforms to the ACCORDANT metamodel to express BDA architectures.
- A set of architectural mismatch definitions expressed as inter-view and intra-view constraints.
- A complete implementation of constraint specifications expressed in OCL and a semantic query language (SPARQL).
- A set of use cases to validate our hypotheses.
- A survey applied to BDA practitioners to evaluate our modeling framework.

1.7 Methodology

This section presents the Design Science Research (DSR) methodology [48] adopted to tackle the research problem presented in Section 1.4 and to achieve the research objectives described in Section 1.5.

Design Science Research (DSR) [48] aims to acquire the understanding of a problem domain and its solution achieved by building and applying designed artifacts (i.e., constructs, models, methods, and instances). These artifacts are evaluated through empirical, qualitative, or quantitative methods. DSR was selected because this research has as objective to solve a problem in the architecture domain, supported on artifacts that should be analyzed, designed, implemented, and evaluated. DSR is commonly used in research related to technologies and information systems.

In this research, we have adopted DSR as detailed in Fig. 1.7.1 adapted from the framework proposed in [48]. DSR defined three main elements: 1) the environment defines the problem space in which reside the phenomena of interest; 2) the information systems (IS) research aims to design, develop and evaluate artifacts using rigorous methodologies to solve the stated problem; 3) the knowledge base that examines the foundation, methodologies, and frameworks to support the definition and design of the solution artifacts.

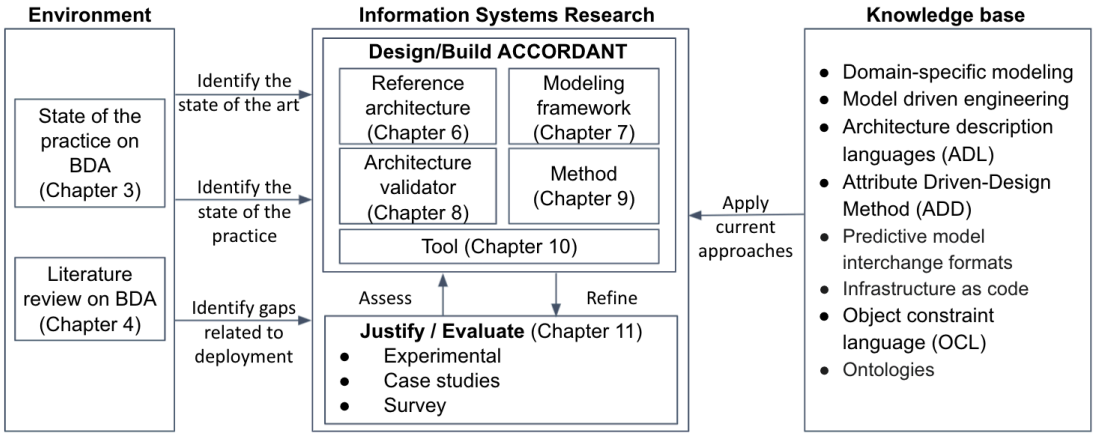

Figure 1.7.1: Design science research methodology adopted in this thesis

Firstly, to establish the environment of this research, we 1) design and apply a survey to identify the procedures, tools, and challenges of BDA in practice, and 2) a systematic literature review to determine state-of-the-art related to BDA deployment. Second, we used existing foundation, methodologies, and frameworks which provide the baseline of our research: ACCORDANT.

Third, we analyzed, designed, and built the ACCORDANT's artifacts considering the environment and knowledge base. To evaluate ACCORDANT's artifacts, we adopted Sonnenberg and Brocke's framework [89] which defines evaluation activities in the DSR process, see Fig. 1.7.2.

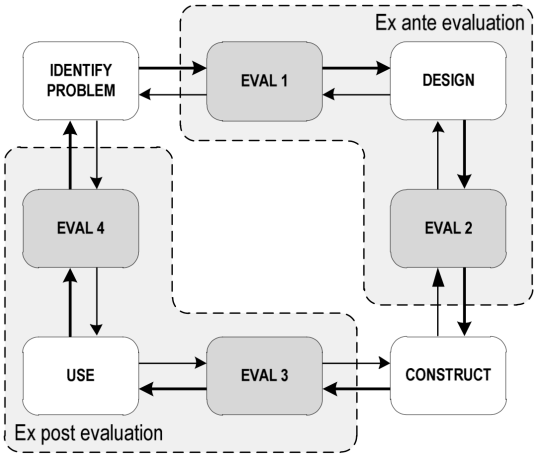

Figure 1.7.2: Evaluation activities within DSR. From [89]

The evaluation activities in DSR considers ex-ante/ex-post and artificial/naturalistic evaluations. Ex-ante evaluations are performed before the artifact construction, and ex-post evaluation occurs after the artifact construction. Artificial evaluation is always empirical and may be interpretative, positive, or critical (e.g., simulations). Naturalistic evaluation validates the performance of an artifact in a real environment (e.g., real scenarios in the industry). We will apply ex-ante, ex-post, and naturalistic evaluations since ACCORDANT is designed to be used by stakeholders to solve architectural needs (i.e., socio-technical artifacts). In this research, we will apply ex-ante (EVAL1 and EVAL2) and ex-post (EVAL3) evaluations detailed in Chapter 11. The evaluation activities proposed in [89] are defined as follows:

- *EVAL1*: this evaluation activity validates the problem identification for ensuring that a relevant DSR problem is selected and formulated. EVAL1 demonstrates that the problem is practical, novel, and represents a research gap using methods such as literature reviews, review practitioner initiatives, expert interviews, or surveys.

- *EVAL2*: the evaluation of the artifact design to validate this design contributes to the solution of the stated problem. EVAL2 demonstrates the artifact's design considering methods such as demonstrations, simulations, or expert interviews, among others.

- *EVAL3*: the evaluation to initially demonstrate whether and how well the artifact works when interacting with users in an artificial setting. EVAL3 validates the design artifact through prototypes, system experiments, or surveys with experts.

- *EVAL4*: this evaluation validates that an artifact is both applicable and useful in practice inside a naturalistic setting. EVAL4 uses methods such as case studies, field experiments, or surveys.

1.8 Document outline

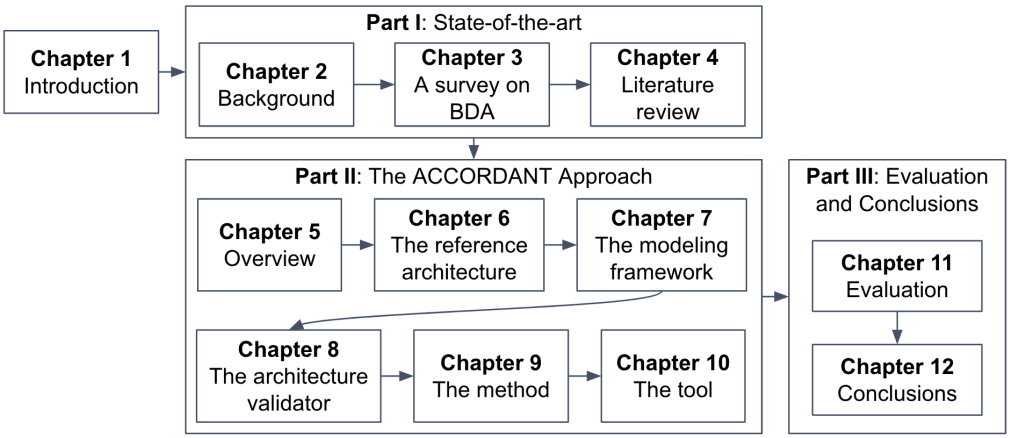

Figure 1.8.1: Document chapters outline

This document is divided into twelve chapters organized into three main parts. Part I is composed of Chapters 2-4 and provides the state-of-the-art scoping background, current practices and challenges, and literature review in the field of BDA deployment. Part II presents our approach in Chapters 5-10. Part III reports the evaluation and conclusions in Chapters 11-12. Figure 1.8.1 describes the structure of this document. The remainder of this document is organized as follows. Chapter 2 offers the conceptual background related to this thesis. Chapter 3 presents our survey results about the state-of-the-practice. Chapter 4 reviews, analyzes, and synthesizes the state-of-the-art through a mapping study. Chapter 5 introduces our approach and describes the main components. Chapter 6 presents a reference architecture for BDA applications. Chapter 7 introduces the ACCORDANT Modeling Framework (MF). Chapter 8 presents the architecture validator. Chapters 9 and 10 detail the ACCORDANT Method and Tool respectively. Chapter 11 includes the evaluation applied to our approach. Finally, Chapter 12 summarizes the conclusions.

Part I

State-of-the-Art

2
Background

This chapter describes the key concepts and practices on which this proposal is based. This background covers the knowledge base of our DSR methodology adoption presented in Section 1.7.

2.1 Domain Specific Modeling (DSM)

Domain-Specific Modeling enables software to be modular and resilient to changes through the separation of concerns principle [57]. The specification of the technology-agnostic concepts, relationships, and constraints within the domain allows users to design and discuss the high-level abstractions that help manage complexity and postpone technology decisions until the implementation stage. Models and code alignment presents different approaches as stated by Kelly and Tolvanen as depicted in Fig. 2.1.1. On the left side, developers code the software product directly without models, which could work for small programs. In the second approach, developers design models to better understand the problem and solution, but the models are only documents separated from actual implementations. In the third case, models can be the result of reverse engineering to extract the high-level concept from already implemented code for visualization purposes. Round-tripping aims to maintain the identical software versions updated

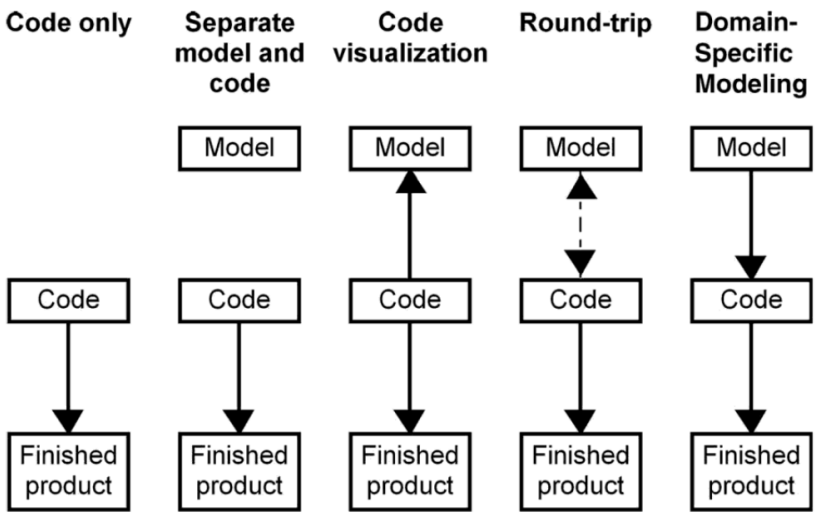

Figure 2.1.1: Aligning approaches of code and models. From [57]

automatically in both places. Finally, in the DSM approach, the models are first-class citizens that allow us to define concepts in the problem domain and generate code from these definitions according to the target technology platform. An essential advantage of DSM is that it closely maps problem and solution domains to provide automation via mappings between models and implementation code. Moreover, DSM compilers can optimize the code generated for the specific platform that helps software engineers increase their productivity.

DSM is supported by the model-driven engineering (MDE) paradigm that emphasizes the use of models as the primary artifact in all phases of the development life-cycle [58]. MDE fosters the automatic generation of system implementations based on its model, either directly or by first transforming the model into a new model that includes the target technology's concrete features. In order to enable the code generations, the domain model must be bounded and conformed to a language specification, the metamodel. Furthermore, the models can be read, checked, validated, and interpreted due to the bounded metamodel's scope. Code generators read the models and translate them into target languages (specific technology). Regarding the representations, DSM can be expressed in graphical, textual, or mixed notation according to the domain context. These notations or languages are called Domain-Specific Languages (DSL). It is possible to embed multiple views or aspects (for example, analytics and deployment) within the domain using different languages and representations that share some elements and relationships.

2.2 Object Constraint Language (OCL)

Object Constraint Language (OCL) is a formal language to define constraints and queries over UML models, metamodels, and stereotypes [99]. OCL was proposed in 1997 by the Open Management Group (OMG) as a standard for object-oriented analysis and design facilities. OCL complements DSM and MDE by adding solidness, consistency, and coherency to the models. OCL is declarative, strongly typed, and side-effect-free, and based on first-order logic that supports quantifiers (for all, exists) and set operations. OCL offers four kinds of expressions to define constraints: i) *invariants* apply to classifiers that must always hold. ii) *preconditions* are constraints that must hold true before an operation execution, whereas iii) *postconditions* must hold true after an operation execution. Furthermore, iv) *guards* are constraints applicable to transitions in a state machine.

2.3 Big data analytics deployment

Analytics methodologies such as Cross Industry Standard Process for Data Mining (CRISP-DM)[24], and ASUM-DM (Analytics Solutions Unified Method) [51] include typically phases of analytics' life cycle: business understanding, data preparation, model building, model evaluation, and deployment. The deployment phase puts the analytics models into production environments. Specifically, this phase determines the architecture and describes how software components will be deployed over technology infrastructure and how the architecture will fulfill the QAs through BDA tactics [43] defined by business operation.

The inputs of the deployment phase are the analytics transformations and models built and validated in the data preparation, model building, and model evaluation phases. The output of the deployment phase is the BDA software solution installed and working in a productive environment to offer the analytics services required by the business operation. The activities involved in the deployment phase to build an analytics implementation comprehends quality scenarios definition, architecture design, software development, testing, software deployment in computational nodes, and monitoring.

Due to current differences between analytics process and software development, the transition of analytics products to software components involves a costly refactoring or adaptation. To offer a smooth transition and interoperability between both domains, open standards such as PMML (Predictive Model Markup Language) [45] and Portable Format for Analytics (PFA) [79] have been proposed to enable transformations and models interchange among several an-

alytics tools. Due to the wide use of these specifications by analytics tool vendors, they are considered de-facto standards for interoperability [45].

2.4 Reference Architectures (RA)

Bass et al. [9] define a reference architecture (AR) as the set of functionalities of a known domain mapped to software elements, together with their information flows. The RAs define good practices and a neutral technology guide for implementing software architectures in a particular domain. They provide a common vocabulary on which to discuss different implementation proposals. Angelov et al. [5] classify RAs based on the following characteristics: i) objective, since their purpose may be to standardize software architectures to favor interoperability or facilitate implementation by providing inspiration or guidelines to design; ii) organizations that use it (a single organization or multiple organizations that are part of the same sector); iii) type of definition, RA is defined as preliminary when the technology, software solutions or algorithms required for its application do not yet exist in practice, while a classic RA is defined when these artifacts exist at the time of their design and are tested in practice; iv) level of detail: detailed, semi-detailed, or aggregated; and v) level of formality: informal, semi-formal, or formal.

2.5 Software architecture design

An architecture description is composed of *architectural views* to address different concerns, and these views are built based on the collection of patterns, templates, and conventions called *Viewpoints*. QS drives the architectural design and functional requirements through a systematic design method, such as ADD [22]), and it could be evaluated using methods such as ATAM [56]. ADD comprises seven steps: 1) Review inputs (purpose, functional requirements, QS, and constraints). 2) In each ADD iteration, a design goal is defined from these inputs. 3) Choose systems elements to refine. 4) Choose design concepts to satisfy the selected drivers. 5) Instantiate architectural elements and define interfaces. 6) Sketch views and record design decisions. 7) Analyze current design, review goal achievement and design purpose, and start a new iteration (from step 2) if selected drivers are not satisfied.

2.6 Architecture trade-off analysis method

The Architecture Trade-off Analysis Method (ATAM) was introduced in [56] to support design trade-offs. The specific purpose of ATAM is to analyze and assess the software architecture in

light of quality attribute requirements. In addition, ATAM aims to detect potential risk areas within the architecture of a complex software-intensive system. The inputs of ATAM include business goals, quality scenarios (QS), and architecture approaches. A QS is a quality-attribute-specific requirement that consists of a source of stimulus, stimulus, environment, artifact, response, and response measure. Architectural approaches such as styles or patterns are means to achieve a QS.

2.7 Architecture Description Languages (ADL)

An Architecture Description Language (ADL) is a DSL in the software architecture domain. An ADL focuses on software architecture modeling to describe architectural elements such as components, properties, constraints, and relationships among them. There is little consensus in the academy on what an ADL is and what aspects should be modeled caused mainly by the different purposes to use the ADL[68]. An ADL can be used for architectural information expression, model instantiation, architectural analysis, evaluation of architecture alternatives, and conformance verification to a particular specification [88]. Despite this, previous ADLs have in common the inclusion of the following modeling features: components, connectors, architectural configurations, and tools support [68].

2.8 Architectural mismatch

The architectural mismatch term was defined by Garlan et al. in [36] as incompatible assumptions that each part makes about its operating environment. These assumptions are often implicit, which hardens analyze and detect them before building the system or when the system evolves, hence its importance for software reusing and composition. According to Garlan et al. [37], the causes of these mismatched assumptions can be categorized in: the nature of the components (including the control model), the nature of the connectors (protocols and data), the global architectural structure, and the construction process. In addition, mismatched assumptions can occur in three facets of component interaction: application domain, infrastructure, and between peer components.

Three basic techniques have been addressed in [37] to deal with architectural mismatch: Prevention, detection, and repair. *Prevention* usually involves architectural specialization and standardization to restrict and regulate components and interactions within specific domains. *Detection* can be supported by documentation standards to make assumptions explicit and archi-

tectural prototyping within the design process to expose architectural mismatch early. *Repair* mismatches in some cases can be the only available option, and these cases require to include mechanisms such as adapters, wrappers, mediators.

2.9 DevOps and Infrastructure as Code (IaC)

According to Bass. et al. [10], DevOps is a set of practices that aims to reduce the time from software development to production environment, ensuring high quality. DevOps includes activities as deploy, operate and monitor applications, with the goals of improve deployment frequency and speed up the time to market what is aligned to our proposal's objectives. Infrastructure as Code (IaC) arises from the necessity to handle the infrastructure setup, evolution, and monitoring in an automated and replicable way through executable specifications [70]. IaC promotes the reduction of cost, time, and risk of IT infrastructure provision by offering languages and tools that specify concrete environments (bare metal servers, virtual machines, operative systems, middleware, and configuration resources) and allocate them automatically. In this context, technologies such as Kubernetes[1], an open source for automating deployment, scaling, and management of container clusters which offer to decouple application containers from the infrastructure details.

2.10 Ontologies

An ontology is an explicit description of a knowledge domain, defined in terms of its concepts, properties, attributes, constraints, and individuals[71]: $O = \{C, P, H^C, H^P, A^O, I, R^I\}$. Where C is the set of concepts, P set of properties, and H^C is the hierarchy of relationships among the concepts, such that $H^C \subset C \times C(c_i, c_j) \in H^C$ represents that concept c_i es is a sub-concept of c_j. H^P defines the hierarchical relationships among object properties, A^O is the set of axioms, and I is the set of individuals, i.e., instances of concepts and properties associated through R^I relational instances. A key advantage of ontologies is to provide useful characteristics for intelligent systems, knowledge representation, and engineering, and there is extensive research to integrate MDE and ontologies to add a reasoning layer on meta-modeling [38].

Different languages specify ontologies, like the Resource Description Framework (RDF) and the Ontology Web Language (OWL). These languages allow designers to express the relationships between things by standardizing on a flexible, triple-based format and then providing a

[1]https://kubernetes.io/

vocabulary such as *rdf:type* or *rdfs:subClassOf* to define the class hierarchy. These languages are XML-based, and they are used to define and publish ontologies on the web through expressive description logic. SPARQL[2] is a query language to retrieve and manipulate data stored in RDF or OWL format. A SPARQL query consists of triple patterns, conjunctions, disjunctions, and optional patterns, which can be resolved using graph pattern matching.

[2]https://www.w3.org/TR/sparql11-query

3

A survey on BDA development and deployment

This chapter[1] presents a survey to identify the practices, techniques, and tools used in the BDA projects to understand the challenges that emerge in this context and obtain the main insights to tackle them.

3.1 Introduction

In Section 1.2, we reviewed studies that have addressed BDA adoption and challenges in analytics practices, which motivate our work, but little research has been carried out to identify practices, behavior, and procedures from the perspective of software engineering and architecture in the BDA. When architecting BDA applications, some QAs become particularly important compared to other domains when it is crucial to scale and be available when processing a large amount of data over distributed infrastructure. Besides, data scientists train and evaluate ML algorithms regarding their accuracy, performance, and interpretability. Hence, it is valuable in this research to know how stakeholders deal with the trade-offs among these quality attributes.

[1]Portion of this chapter previously appeared in Castellanos C., Pérez B., Varela C.A., Villamil M..P., Correal D. (2019) "A Survey on Big Data Analytics Solutions Deployment". In: Bures T., Duchien L., Inverardi P. (eds) Software Architecture. ECSA 2019. Lecture Notes in Computer Science, vol 11681. Springer, Cham

The aforementioned aspects drive the development of a survey to identify practices, techniques, and tools used in designing, developing, and deploying BDA projects from a software architecture perspective. This purpose is aligned to the research objective *RO1* stated in Section 1.5.

According to Fink [35], a survey is a comprehensive research method for collecting information to describe, compare or explain knowledge, attitudes, and behavior. We conducted a survey among practitioners following a methodology proposed by Kitchenham et al. in [59]. We collected answers from 76 practitioners engaged with cross-industry BDA projects in Colombia. This survey aims at the following research objectives framed in the BDA development and deployment context:

- To determine the practices and methods followed by practitioners.
- To identify techniques and tools used in practice.
- To identify the perceived challenges when developing these projects in the industry.
- To identify the main quality attributes considered throughout the solution design and deployment.

The remainder of this Chapter is structured as follows: Section 3.2 describes our research methodology. Sections 3.3 presents the survey results. Section 3.4 discusses the findings. Section 3.5 presents the threats to validity. Finally, Section 3.6 summarizes this chapter.

3.2 Methodology

To conduct our study, we follow the methodology proposed by Kitchenham and Pfleeger [59] for survey design in empirical software engineering. This methodology proposes a set of activities needed to undertake a survey: setting the objectives, survey design, developing the survey instrument, evaluating the survey instrument, obtaining valid data, and analyzing the data. Following the proposed methodology, we formulate the research questions based on the objectives presented in Section 3.1.

3.2.1 Research questions

We formulate the research questions (RQs) of this survey based on the objectives presented in Section 3.1.

RQ1: *What are the practices, methods, techniques, and tools used in BDA development and deployment?* By answering this question, we intend to characterize practices, techniques, and tools used in BDA design, development, deployment, and operation.

RQ2: *What are the main challenges faced in BDA development and deployment?* By answering this question, we aim at identifying the challenges practitioners have to face in this context.

RQ3: *What are the main quality attributes considered in BDA modeling, evaluation, and deployment stages?* By answering this question, we aim at characterizing QAs which drive BDA's software architecture.

3.2.2 Sample and population

In our survey, the target population entails practitioners who have participated in BDA projects playing roles such as project manager, business expert, requirements engineer, data scientist/-analyst, data engineer software designer/developer, or IT architect. We employed *Convenience sampling* (a non-probabilistic sampling method [59]) for selecting the population because of our access to participants involved in BDA projects. Participants were available through the master programs in Information Engineering and IT Architecture offered by Universidad de Los Andes, and the Colombian Center of Excellence and Appropriation in Big Data Analytics (CAOBA). These participants were involved in industry BDA projects, and they were available to collaborate in this research. Master students were enrolled in IT Architecture and Data Science Applied courses.

Inclusion and exclusion criteria enable us to choose valid answers regarding experience in BDA and consistency. This survey considered the following Inclusion criteria: (i) The respondent has industrial experience in BDA projects, and (ii) The respondent has academic experience in BDA projects. The exclusion criteria were (i) There are inconsistent answers (i.e., self-contradictory), and (ii) respondents that answered less than 50% of the questions.

3.2.3 Survey design

This survey is classified as descriptive research because: 1) This survey was preplanned and structured, and 2) the information collected can be statistically inferred over a population. This type of research uses closed-ended questions to understand opinions or attitudes by a group of people on a specific topic. This survey is a self-administered questionnaire, where a research participant is given a set of questions to answer via a paper-based and web-based questionnaire. We included

an opening paragraph to introduces the purpose, concepts, and considerations needed to answer the instrument.

3.2.4 Developing of survey instrument

Our questionnaire consisted of 5 parts and 24 questions as presented in Figure 3.2.1 written in Spanish, the participant's native language. Eighteen questions corresponded to closed-ended and single-choice questions, and seven questions included multiple-choice grids to specify the respondent's level of agreement or disagreement on a Likert scale. All questions were mandatory. The five parts of the survey were: (a) demographic questions, (b) questions about practices, behavior, and challenges in BDA context, (c) questions about techniques and tools used in BDA projects, (d) questions about BDA deployment, and (e) questions about how practitioners dealt with quality attributes. Figure 3.2.1 also details how each questionnaire's part is related to the Research Questions (RQ).

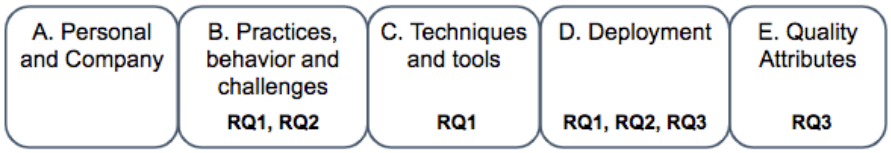

Figure 3.2.1: Questionnaire sections and research questions

Demographic questions asked for job, role, level of education, and experience of the subjects. These questions also asked for company information like industry sector, size, experience, and maturity. This first section helped us to understand the participant's background. The remaining parts were used to collect data about the general perception of deployment of BDA projects.

3.2.5 Evaluating the survey instrument

The questionnaire was reviewed externally by two other researchers, and they checked the content, meaning, and understandability. Additionally, nine practitioners on BDA projects answered a pilot to refine the instrument and estimate the time needed to complete the survey.

3.2.6 Data analysis

Data analysis was done through the following steps: (i) collection of responses into a single spreadsheet, (ii) analysis of the spreadsheet using descriptive statistics for quantitative answers

for each given response, and (iii) identification of key findings from results of the statistical analyses. In order to enable the full replication of this research, a package with the questionnaire and raw answers is publicly available[2].

3.3 Survey results

This section reports the survey results based on collected data, and the following four subsections address the questionnaire's sections detailed in Fig. 3.2.1.

In total, 115 answers were collected, of which 39 (33.9%) were excluded by criteria detailed in Section 3.2.2. The remaining 76 (66.1%) valid answers were further analyzed. Hereinafter, the 76 subjects who respond with valid answers are denominated "respondents".

3.3.1 Personal and company data

This subsection describes the background information of the respondents. This background can influence the perspective and perception of the BDA development and deployment process. This information includes the respondent's profession, educational background, the role played, and experience in BDA projects.

Regarding respondent's professions, the vast majority of them (84.2%) are IT professionals, followed by mathematicians/statistics (5.2%), engineers Non-IT, and business administrators (3.9%). The respondent's role in BDA allows us to classify the stakeholder groups introduced in Section 8.1: IT managers corresponds to 26.3%, software architects: 19.7%, developers: 15.7%, data scientists: 14.4%, and IT operators: 6.5%. We also asked respondents about their level of education. Most of them (40.7%) holds an M.Sc degree, 35.5% have a B.Sc. degree, 22.3% a specialization degree, and one respondent holds a Ph.D. degree.

The question related to work experience in BDA projects shows that most of the respondents are junior-level; hence 67.1% have got involved between 1 and 2 projects, 22.3% have participated between 3 and 5, and 10.5% above five projects. Regarding the years of experience, half of the respondents have worked between 1 and 3 years, 32.8% less than one year, and 10.5% between 3 and 6 years. Finally, 6.5 percent of the participants have six years of experience or more.

We asked the company's sector of the respondents to understand the business environment in which BDA projects are developed, and education (23.6%) is the most common sector. Technology is the second-most common sector with 22.3%. Both Financial and Government sectors

[2]https://storage.cloud.google.com/ccastellanos/BDA-Survey-package.zip

are in the third place with 13.1% of participation, while Communication (9.2%) and Transport (5.2%) sectors complete the top six list.

Questions 8 and 9 inquire about the company size and experience by measuring the number of employees and projects undertaken within the company. Most respondents (63.1%) work in large companies (more than 250 employees), 18.4 in small (between 11 and 50), 13.1% in medium (between 51 and 250 employees), and only the 5.2% in micro-enterprises (less than 11 employees). About the number of BDA projects, 47.3% of participants work in companies with 1-to-3 projects, 15.7% in companies with more than nine projects, and 14.4% in companies with 4-to-6 projects. Finally, 5.2% of respondents answer that their companies have not developed BDA projects, and 2.6% work in companies with 7-9 projects.

To know the level of BDA appropriation in companies, we asked about the current status of BDA projects. As a result, pilot projects were reported in progress by 32.8% of respondents, 23.6% have at least an active program in production, 17.1% in exploration, 9.2% have no plan, and 5.2% have a defined plan to be implemented.

3.3.2 Practices, behavior, and challenges

Fig. 3.3.1 depicts the perception of collaboration and teamwork among the stakeholders involved in the BDA environment. This perception is scored from 1 to 5 (1-Difficult and disjointed and 5-Very fluid and articulated). Analytics and IT collaboration and teamwork have the best perception with a positive scoring (greater than 3) for 56.5% of the respondents. Business/IT and Business/Analytics interactions report the worst collaboration and teamwork perception with only 26.3% and 22.3% of positive scoring, respectively.

Figure 3.3.1: Collaboration and Teamwork.

We also inquired about the difficulty of carrying out each BDA phase to identify the most challenging activities in the BDA life cycle regarding traditional methodologies [24, 51]. This difficulty score ranges from 1 to 10, and the results are presented in Fig. 3.3.2 as boxplot graphs,

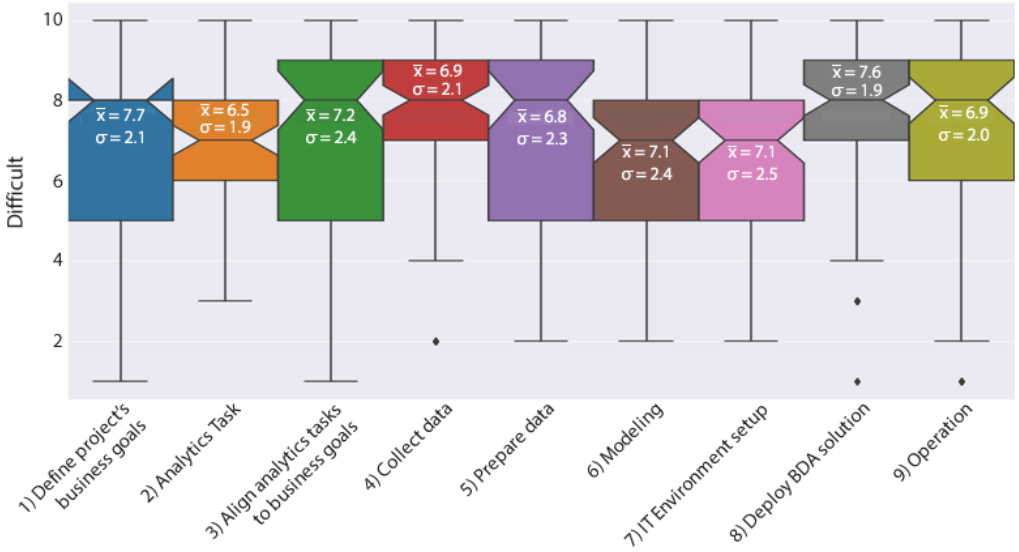

Figure 3.3.2: Level of difficulty to perform BDA activities

including mean ($\overline{x}$) and standard deviation (σ). Six out nine activities observe the highest medians (8 points of difficulty): *1) Define project's business goals, 3) Align analytics tasks to business goals, 4) Collect data, 5) Prepare data, 8) Deploy BDA solution* and *9) Operation.* Among these six activities, those that present the highest difficulty means are: *1) Define project's business goals* ($\overline{x}$=7.7, σ=2.1), *3) Align analytics tasks to business goals* ($\overline{x}$=7.2, σ=2.4), and *8) Deploy BDA solution* ($\overline{x}$=7.6, σ=1.9). The boxplots of these three challenging activities show that *8) Deploy BDA solution* activity has the smallest interquartile range (between 7 and 9), while the other two activities show a higher dispersion. It implies that Deployment activity exhibits both the highest average of difficulty and the lowest variance scoring among respondents.

3.3.3 Techniques and tools

We asked respondents to categorize the usage of an arrangement of techniques to know how data scientists deal with and work with many options. Fig. 3.3.3b describes the frequency of use of analytics techniques/algorithms to build analytics models on a scale from 1 (rarely used) to 5 (frequently used). The five most prevalent techniques are, in descending order: aggregations, regression, clustering, anomaly detection, and principal component analysis (PCA). Aggregations could be the most simple techniques, but they are the most used when data analysis is required. Most novelty techniques such as deep learning and support vector machines (SVM) present the lowest usage level in the respondents' context.

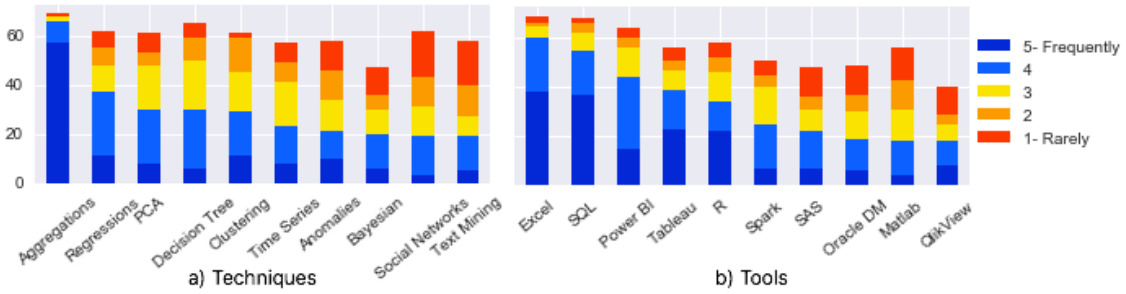

Figure 3.3.3: Top-ten of usage level of a) Techniques and b) Technology Tools

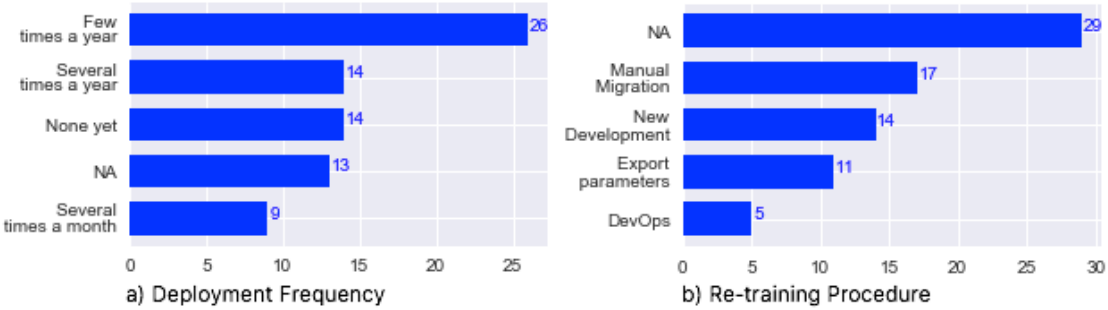

Figure 3.3.4: Frequency of a) BDA deployment in productive environment and b) Re-training Procedure.

In addition to the techniques, we also asked about technology tools usage in BDA development through the same scale from 1 to 5, and Fig. 3.3.3b summarizes the results obtained. It is worth noting that this question comprised a wide range and kind of tools, from spreadsheets to distributed processing engines, including self-service Business Intelligence (BI) tools. Excel and Standard Query Language SQL are the tools most frequently used tools with 78.9% and 72.3% high rate usage, respectively. The following eight most used technologies are in descending order: Power BI, Tableau, R, Click view, Spark, SAS, IBM SPSS, and Oracle Data Mining. Commercial and visual-supported tools such as Microsoft Power BI, SAS, IBM SPSS, OracleDM, and QlikView dominate the top ten of the tool preferences. In contrast, open-source tools and code-driven tools are not widely used, with the exception of R and Apache Spark.

3.3.4 Deployment

In Fig. 3.3.4a, the frequency of BDA deployments on a production environment is shown. As can be noted, few times a year (34.2%), several times a year (18.4%), and "None yet" (18.4%) are the predominant answers, thus confirming the low deployment frequency in our study's context.

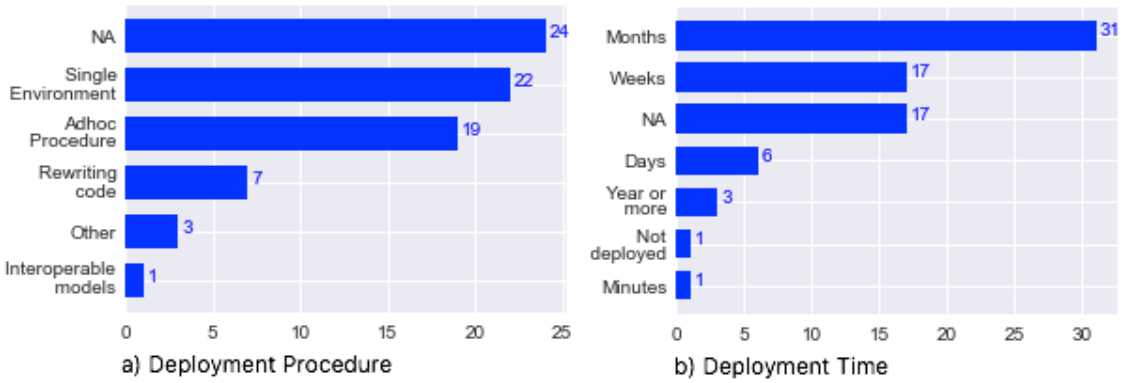

Figure 3.3.5: Frequency of a) Deployment Procedure and b) Deployment Time.

During maintenance and operation stages is necessary to retrain/adjust models and software to have up-to-date services. Fig. 3.3.4b depicts the procedures used to do such retraining. 22.3% of respondents retrain the model in data lab environments, upgrading the production model via manual migration. The second group of respondents reports that they do not retrain models, but they develop a new code version (18.4%). The 14.4% of respondents train the model and export new parameters to production, and only the 6.5% use a DevOps approach.

The respondents were consulted about the procedure or methodology to package/migrate the analytics models and data transformations from the data lab to production, and Fig. 3.3.5a shows these results. Noteworthy, 31.5% of the respondents did not know or answer which deployment procedure is used. The 28,9% of respondents reported they do not have a procedure because they have a single environment of BDA, use an ad-hoc procedure (25%), or rewrite the whole source code (9.2%). Only one respondent (1.3%) reported the use of interoperable models such as PMML or PFA.

To gain first-hand knowledge about the lag time in the deployment of BDA solutions, we also inquire about the time elapsed between model development and its deployment in production. Fig. 3.3.5b details the time scales invested in deployments. The most common time scale is *months* (40.7%), followed by *weeks* (22.3%), and in a lower proportion, *days* (7.8%).

To understand the relationship between deployment procedure and frequency, we compare these questions' results in Fig. 3.3.6. Ad hoc procedure is the most common for monthly (44.4%, 4 out of 9) and yearly deployments (42.3%, 11 out of 26). Although maintaining a single environment is used in projects with several deployments a year (35.7%, 5 out of 14), a single environment is the most common approach in projects which has not yet been deployed. Specifications for inter-operable predictive models are not used or scarcely used, displaying a lack of

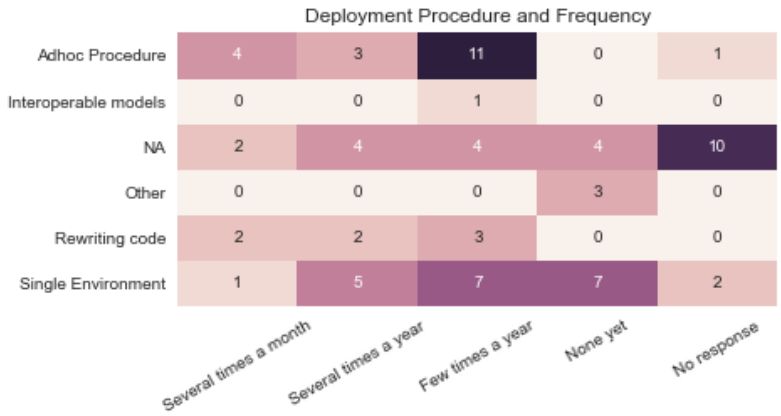

Figure 3.3.6: Deployment Procedure/methodology and Frequency.

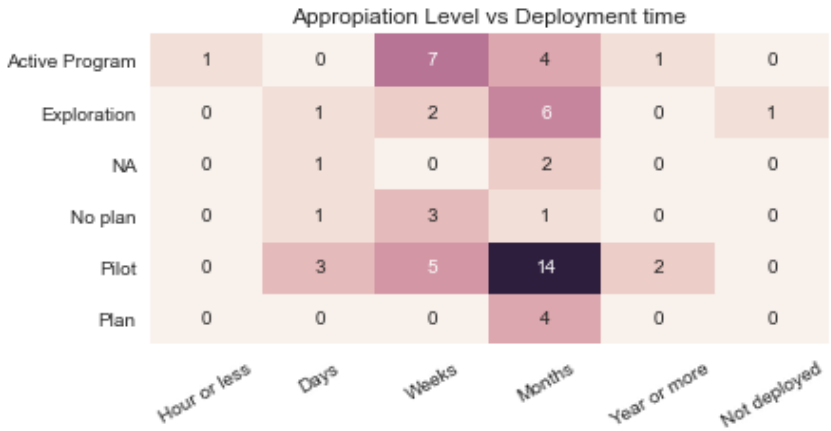

Figure 3.3.7: Appropriation Level and Deployment Time.

knowledge about these standards.

Figure 3.3.7 compares the appropriation level of the company with the deployment time. Companies with active BDA programs take weeks 46.6% (7 out of 15) and months 24.6% (4 out of 11) to deploy their solutions. Most of the organizations with a BDA plan (100%), pilot project (58.3%), and exploration phase (60%) take months to deploy their applications. Comparing deployment procedures and deployment procedures depicted in Figure 3.3.8, it is noticeable that companies with active programs use mainly (50%) ad hoc procedures. Similarly, companies with pilot projects use ad hoc procedures 28% (7 of 25), no-answer 28% (7 of 25), and code rewriting 20% (5 out of 25). Finally, most of the projects in the exploration phase (53%) or without a BDA plan (71.4%) use a single environment approach (i.e., data lab and production

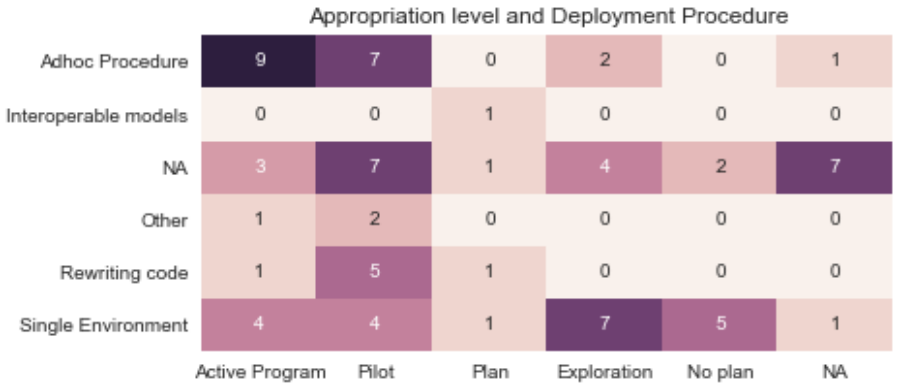

Figure 3.3.8: Appropriation Level and Deployment Procedure.

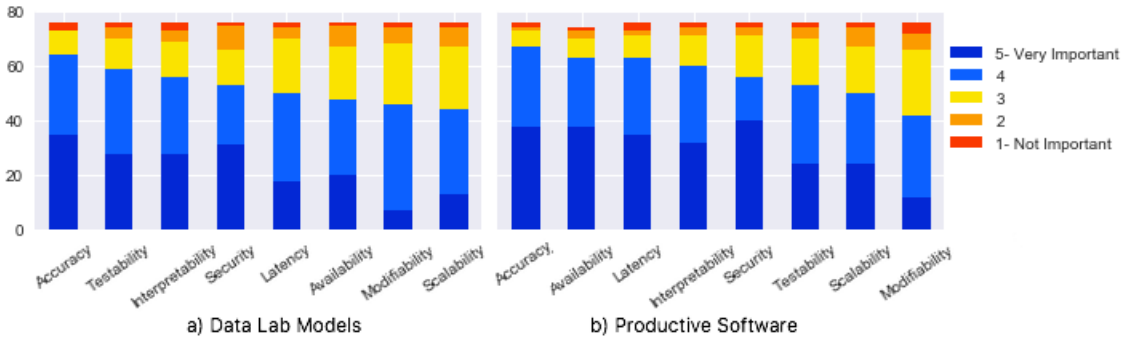

Figure 3.3.9: Quality Attributes Relevance in the a) Data Lab and b) Productive Software Solution.

are the same environment).

3.3.5 Quality attributes

The architecture design implies prioritizing some quality attributes over others to achieve expected outputs. For this reason, we analyze a set of questions oriented to answer RQ1.

Fig. 3.3.9a details the weights of relevance (from 1 to 5) for each QA during the analytics development in the data lab environment. The most relevant QA is accuracy, with 84.2% of positive ratings (i.e., greater than 3), followed by testability (77.6%), interpretability (73.6%), security (69.7%), and response time (65.7%). Availability and scalability observe the lowest importance ratings (63.1%, 60.5%, and 57.8%, respectively) of relevance inside the data lab.

On the other hand, the same question about QA's relevance was asked, but in the production environment to compare the quality priorities. Fig. 3.3.9b shows that accuracy continues in the first place with 88.1% of respondent's positive ratings. The second and third places are occupied

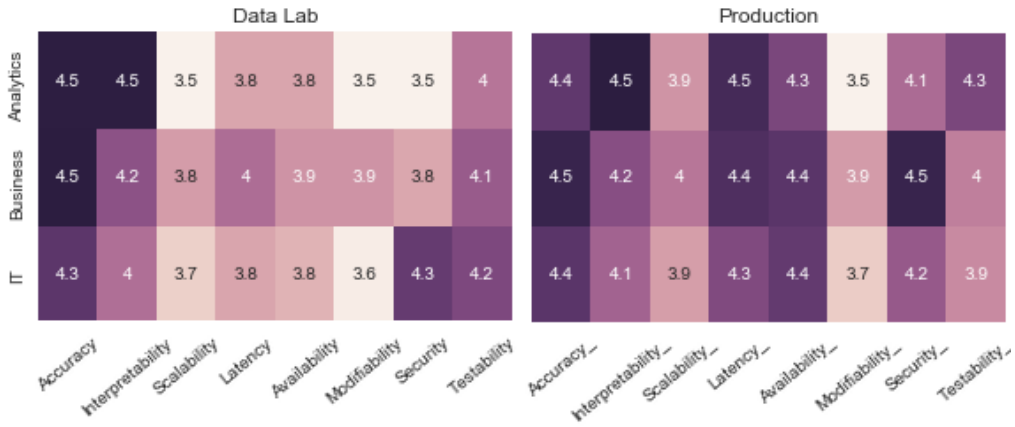

Figure 3.3.10: Quality attributes relevance regarding stakeholder domains.

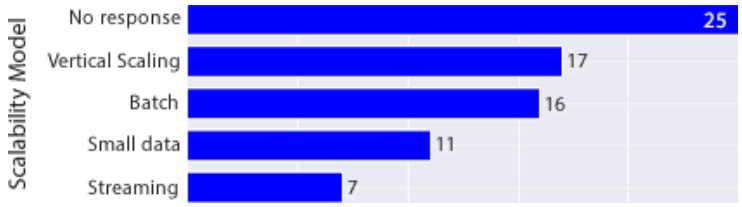

Figure 3.3.11: Scaling approaches.

by performance QAs: availability (82.8%) and response time (82.8%). Interpretability falls to fourth place with 78.9% of positive ratings, and security ends the top 5 list with 73.6%. Despite the fact that scalability and modifiability maintain the last two places (65.7% and 55.2% respectively), it is worth noting that scalability almost doubles its *Very important* rating from 17.1% to 31.5%.

Fig. 3.3.10 reports QA relevance averages (from 1-Not Important to 5-Very Important) in the data lab and production regarding the stakeholder domains. In the data lab, accuracy observes the highest relevance for all stakeholders with slight differences in magnitude. Data scientists and business stakeholders rank interpretability and testability in second and third place, respectively. While IT stakeholders prioritize security and testability in second and third place, respectively. In the production environment, the picture changes significantly. Data scientists give higher relevance to interpretability and latency than given in the data lab, while business users prioritize accuracy and security. Besides, IT stakeholders rate accuracy and availability with the highest scores. Comparing the relevance scores between data lab and production, the differences in latency, availability, scalability, and security for all stakeholders are remarkable, evidencing an

evident change of QA priorities between environments.

Finally, we included a question to know how respondents achieve scalability in the BDA context. Fig. 3.3.11 summarizes the respondent's answers. The most noticeable result is that most of the respondents do not know or do not respond (32.8%, 25 out of 76), which could reflect the lack of knowledge or interest about the technical capabilities to support big data processing. Vertical scaling is based on adding more resources to a single machine, and this was the most used approach with 22.3% of answers. 21.1% of respondents report distributed batch processing using big data frameworks such as Hadoop or Spark. 14.4% of respondents declare they do not require scaling capabilities because they work with small data. Only 9.2% of respondents require distributed streaming processing.

3.4 Discussion

The BDA adoption and appropriation among companies is incipient, as shown by results in which 47% have only developed between 1 and 3 projects, and only 23.6% have an active BDA program. This situation is slightly better than reported by the Colombian IT Ministry [55] that calculates the adoption of big data technologies of 16.8% in big enterprises. Compared to a previous worldwide report in 2016 [82], our survey reports better levels of appropriation in terms of the proportion of active programs in organizations (23.6% versus 17%), pilot programs (32.8% versus 17%) and "no–BDA plans" (9.2% versus 23%). In contrast, we find lower indicators regarding organizations in phases of exploration (17.1% versus 32%) and plans to be implemented (5.2% versus 11%). These results could suggest a growing interest in companies for BDA adoption and their respective progress over time.

This survey found that classic analytics techniques such as aggregations, regression, and clustering are the most used by companies. These results are similar to previous studies [81, 82]; the only exception is that the decision tree model is not ranked in the top three of the most used algorithms in our survey. The most basic tools like Excel and SQL scripts are in the first places, followed by Tableau and R. These preferences are different from specific data science studies where R, SPSS, SAS, and Tableau occupied the top positions. These results can denote unfamiliarity or lack of skills in data science-oriented tools in the Colombian context. Also, visual-assisted tools are more used than code-driven frameworks, suggesting that respondents prefer high-level tools instead of low-level programming languages to build BDA applications. This survey also reports a lack of standard procedures to deploy and operate BDA solutions which frequently implies manual code rewriting and configuration, confirming findings presented previously in [31]. It

is noticeable the lack of knowledge and use of de-facto standards (1.3%) for sharing analytics models across technologies (such as PMML or PFA) compared to previous studies (19%) such as [31], which can promote the cumbersome and delayed process of putting analytics services in operation. These findings allow us to argue that DevOps practices in these specific domains are still unknown, immature, or under-used, and some recent works such as [16, 63] have addressed this concern.

Activities involved in BDA development, such as business objectives and analytics goals definition, data collection, and deployment, are considered "hard" on average. Specifically, deployment seems challenging, probably due to different factors such as software development driven by competing QAs in different environments, tool heterogeneity, and the lack of mature deployment procedures, even in organizations with active BDA programs. These factors have also been identified in previous works [31, 82]. Teamwork and collaboration between data scientists and IT stakeholders are better ranked than business/IT and business/data scientist interaction.

In terms of deployment challenges, our results confirm issues in different facets. A scarcity of deployments into production leads to low operationalization of BDA solutions and prolonged deployment delays, ranging from weeks to months (63%). This scenario can be caused by technical reasons such as inadequate tools and inadequate procedures to deploy and retrain BDA solutions in production environments. These findings coincide with conclusions reported in [82] and [31] where they reported low rates of deployment, lack of procedures to deploy BDA solutions, and long deployment times. Even companies in more mature BDA stages (i.e., active programs) reported deployment times from weeks to months.

The relevant QAs for data analytics modeling are not the same as those during the software development phase. The reason for this is that both artifacts (models and software) pursue different objectives. While the analytics model's quality is measured by the accuracy, interpretability, and testability; the BDA application must achieve expected performance, availability, and scalability. This situation can lead to competing drivers when the software architect makes decisions (i.e., patterns, tactics, technologies) that are not suited for analytics solutions in different environments. These competing drivers could also lead to heterogeneity of technology tools reported along the BDA life cycle.

3.5 Threats to validity

In our study, the research methodology was validated to avoid biases as much as possible. In the following, construct validity, internal validity, external validity, and reliability are presented

together with their mitigation strategies as suggested by Runeson and Martin in [84].

Construct validity. It reflects the relation between operational measures studied and the researcher's main idea, according to the research questions [84]. The phrasing used in sentences for closed-ended questions could be the most recurrent threat in questionnaire-based surveys. In order to mitigate this threat, we first piloted the survey internally several times. Then, we piloted the survey externally with practitioners involved in BDA projects through an online survey to refine the used language.

Another risk is related to participants did not find any suitable response in the set of available ones. For this, we included an "Other" answer for each question. In our results, we had a relatively low number of respondents using this alternative answer.

Internal validity. It reflects the presence of causal relations affecting the investigated factor [84]. We performed the data analysis using basic descriptive statistics and performed a cross-analysis of each participant's responses. We also provided definitions used consistently in the survey, allowing the respondents to understand the questions fully.

External validity. It reflects the possibility of generalizing the findings and discovering if the findings are of interest outside the investigated case [84]. For our study, a potential threat refers to the demographic distribution of response samples. We applied Convenience sampling to helped us in selecting study participants. However, we are aware that this sampling technique could decrease the respondents' size. To mitigate this potential threat, we ensure that the respondents' set was a heterogeneous sample regarding demographic information, such as professional experience, educational background, the number of projects.

Reliability. It reflects the independence between the extracted data and the obtained results [84]. To mitigate this threat, we employed observer triangulation, having all authors participating in the data extraction and analysis processes. Due to the non-statistical nature of convenience sampling used in this study, we cannot make strong inferences. We also avoid performing any statistical correlation analysis because we recognize our sample size is small and centered on practitioners who have participated in BDA projects. Despite this fact, our results can open new discussions and research lines.

3.6 Summary

We have presented an empirical study of how practitioners deal with the development and deployment of BDA solutions. We first developed and evaluated a pilot to design a paper-based survey. The data extracted from the questionnaires provide clues for understanding the activi-

ties, behavior, practices, and challenges practitioners face.

We evidenced that the definition of the project's business goals, alignment between business goals and analytics task, and solution deployment were reported as the most challenging activities in the BDA life cycle. We found communication and interoperability concerns across knowledge domains within the BDA life cycle. Our results found competing QAs in different development stages of BDA applications. Concerning the deployment process, we argued that heterogeneity of technology tools, low adoption of interoperable models, and immature or little-known deployment procedures could lead to delayed and sporadic deployments. Simple and visual-assisted tools showed a broader usage than sophisticated code-driven frameworks. Most of the companies' deployment procedures of BDA applications fell short of applying DevOps approaches to automate building, integration, and testing.

Our results offer insights into how to design and implement BDA solutions in the practice of software architecture regarding the related challenges, procedures, and deployment barriers. In addition, the most common methodologies, techniques, and tools in the industry could be a starting point to define a BDA adoption road map. This research motivated us to consider separation-of-concerns among the knowledge domains to reduce the deployment gap by integrating and interoperating business, analytics, and IT.

4

Literature review

In Chapter 3, we analyzed communication, tools, procedures, and challenges in practice during the development and deployment stages. Instead, this chapter aims to review the literature in available in BDA by collecting, synthesizing, and evaluating primary studies to cover the research objective *RO2*.

4.1 Introduction

To achieve the research objective *RO2* introduced in Section 1.5, it is necessary to review the state-of-the-art of BDA development and deployment. We developed a mapping study during the definition of our work in 2018 to summarize the BDA deployment approaches available at that time. We have updated the results of this review with recent studies in the same line.

4.2 A mapping study in BDA design and deployment

The reason to do this systematically arises from the need to summarize the available approaches about the BDA deployment phenomenon in a comprehensive, replicable, and unbiased manner

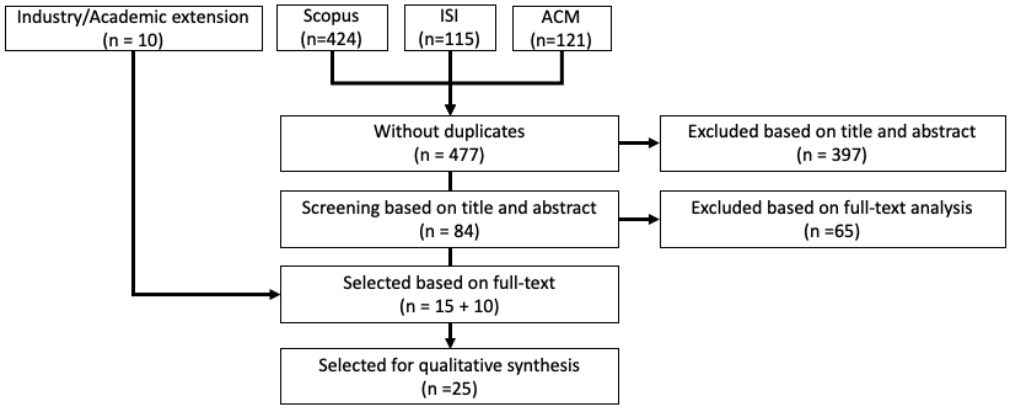

Figure 4.3.1: Selection process of primary studies.

[8]. A mapping study aims to gather all existing evidence, but it also supports the development of evidence-based guidelines for practitioners [60].

As far as we know, there are no systematic reviews about BDA deployments and software architectural aspects like design, quality scenarios, quality attributes, architectural tactics, and patterns. Therefore, we conducted a mapping study on BDA deployment, finding 19 primary studies from 2008 to 2018 and adding seven recent studies to update this research.

The rest of the chapter is organized as follows. Section 4.3 details the followed method. Section 4.4 presents the results of this study. Finally, Section 4.6 summarizes the chapter.

4.3 Method

This study follows the guidelines proposed by Petersen et al. [78], and Kitchenham and Charters in [8]. This section details the execution of steps included in the guidelines. Fig. 4.3.1 depicts the consecutive filters applied to obtain the primary studies.

4.3.1 Research questions

The first step is to define the research question considering the purpose and scope. To address the research objectives previously presented in Section 1.5, we formulate the specific research questions framed for this mapping study. These RQ and their rationale are detailed in Table 4.3.1.

Table 4.3.1: Research questions

Id	Research question	Rationale
RQ1	What are the frameworks, methods, techniques and tools used to design, develop and deploy BDA solutions?	To find out all previous approaches to support the design, development, deployment and operations of BDA solutions.
RQ2	Which analytics life cycle's phases have been addressed?	To identify the coverage in terms of stages of analytics' life cycle introduced in Section 2.3.
RQ3	How are addressed architectural concepts such as QS, tactics and patterns?	To obtain a detailed understanding of the application of software architecture practices in these approaches.

4.3.2 Study search

To perform a systematic search and collection of potentially relevant papers from three available electronic databases, we designed and refined a *query search* iteratively. This query aims to ensure the identification and selection of relevant papers aligned to our RQs; furthermore, it enables the replicability of the search. The query search is detailed in Listing 4.1, and it is composed of three parts separated with *AND* Boolean operators to filter the main subjects covered. These subjects are big data analytics, deployment, and software architecture. For each subject, a set of synonyms and related terms are included with *OR* operators to produce a more comprehensive search. The query string is helpful to perform automatic searching in most electronic databases by matching queries with papers' titles, abstracts, and keywords. However, it is necessary to translate the query into each specific engine's syntax.

We searched three renowned electronic databases: ISI Web of Science, Scopus, and ACM Digital Library. IEEE Xplore, SpringerLink, and ScienceDirect were not included since Scopus, and ISI Web of Science contains the works published in those databases. We chose 2008 as the initial year of the search period, and we executed the search in April 2018. In addition, we have incorporated nine additional studies which have emerged from further reviews after the initial search or from new results obtained by the same research projects identified initially.

The deployment gap phenomenon impacts productivity, and it is of great interest in industry research. To ensure a broader and more thorough searching process, we extended the search sources with industry papers or products relevant to this phenomenon. This review extension produced four industry/academy sources to be included in screening, evaluation, and synthesiz-

```
(data mining OR big data OR analytic* OR
data science OR data intensive) AND
(deploy* OR devops OR infrastruct*) AND
(software engineer* OR solution architect* OR
system architect* OR software develop* OR software architect*)
```

Listing 4.1: Query Search string

ing steps.

4.3.3 Inclusion and exclusion criteria

To carry out a more objective study selection, we define a set of selection criteria employed in each step of the filtering activity presented in Figure 4.3.1. Both inclusion and exclusion criteria (IC and EC respectively) are reported in Table 4.3.2.

Table 4.3.2: Inclusion and exclusion criteria

Id	Inclusion Criteria
IC1	A study which presents guidelines, method, framework or tools to design, develop, or deploy BDA solution regarding software engineering practices
IC2	A study that is peer-reviewed, i.e., published in journals, proceedings or book chapters.
Id	**Exclusion Criteria**
EC1	A study not written in English.
EC2	A study which is published as abstract, tutorial, or talk form.
EC3	A study limited to offers challenges, issues, opportunities or insights in BDA, but it does not include a proposal to address them.
EC4	A study focused on applying ML techniques in a specific field, but it does not address the system architecture.
EC5	A study that presents a software or system architecture for a specific BDA context, but it is not proposed to be generalized.

4.3.4 Quality assessment

Quality criteria allow us to assess the quality of primary studies in terms of the extent to which the study reduces bias and increases the validity by using the evidence hierarchy proposed by Alves et al. in [3]. Additionally, the quality assessment helps to identify the limitations of each study during data synthesis. We assess the extent of use /application of principles/practices of computer science and software engineering. The quality criteria (QCs) listed in Table 4.3.3 have scoring to be assigned to each study.

Table 4.3.3: Quality criteria

Id	Quality Criteria	Scoring
QC1	How does the evidence support the claims?	No evidence (0), demonstration or toy example (1), expert opinions or observations (2), academic studies or controlled lab experiments (3), industrial studies or causal case studies (4), and evidence obtained from industrial practice (5).
QC2	Is it considered the separation of concerns principle?	Yes/No
QC3	Is a technology-neutral approach?	Yes/No.
QC4	Is a cross-industry approach?	Yes/No.
QC5	Which is the coverage of analytics life cycle's phases?	Business goals, ingestion, preparation, analytics, deployment and operation from "No" (0), low (1) to high (5).
QC6	Are the software architecture/engineering practices addressed?	quality scenarios (QS), Patterns/Tactics, and DevOps in a scale from "No" (0), low (1) to high (5).
QC7	Does the approach support architecture validation/verification?	Yes/No

4.3.5 Data collection

To answer RQs stated in Section 4.3.1, we applied the search query, export *bibtext* files, and collect the works' metadata: authors, title, year, and abstract. This data was imported to a spreadsheet to facilitate the reviewing process. They were explained and discussed with two other researchers to guarantee an objective application of inclusion, exclusion, and quality criteria. Then, a testing group performed a second review on the evaluation criteria. Finally, selection and evaluation results were peer-reviewed by at least two researchers to prevent mistakes and bias. The data extracted from each study were:

- Research objective or questions.
- Study summary.
- Study's product/output (methodology, guidelines, model, and software) addressing the research question RQ1.
- Analytics life cycle's phases covered (i.e., business objectives, ingestion, preparation, modeling, deployment, and operation) to answer the question RQ2 and validate QC5.
- Software engineering/architecture practices involved to answer question RQ3 and to evaluate QC6: QS, tactics/patterns, monitoring, and self–adaptation
- Type of validation (QC1).
- Usage of separation of concerns principle (QC2).
- Is the approach a technology-neutral approach (QC3).
- Is the approach a cross-industry approach (QC4).

4.3.6 Data analysis

We extracted the data in the structure detailed in Section 4.3.5. After the selection process, the primary studies were synthesized regarding the quality criteria stated before. Then, we made a quantitative analysis using descriptive statistics and frequency analysis of the RQs predefined. The analysis was focused on understanding the quality of published studies, mainly how they deal with the research questions presented in Section 4.3.1.

4.4 Results

We followed the process depicted in Figure 4.3.1 which reports the works filtered and selected in each step. The query was executed in three digital libraries returning 660 studies: 424 from Scopus, 115 from ISI, and 121 from ACM. Then, we found 397 duplicate papers to be removed, resulting in 477 different studies. Over these papers, we performed the first screening by title and abstract validating inclusion and exclusion criteria detailed in Table 4.3.2. As a result, 397 works were excluded, and 80 works passed to the next filter. Then, we made a more in-depth review of the full text, and we excluded 65 papers considering inclusion and exclusion criteria again. As a result, fifteen primary studies were selected for evaluation, analyzes, and synthesis. As mentioned earlier in 4.3.2, we included ten industry/academy products in this stage: S2, S5, S18, S19, S20, S21, S22, S23, S24, and S25.

Among the primary studies resulting from the process, we found two main research lines. The first research line addresses mainly the functional perspective of BDA applications, while the second research line deals with the underlying architecture and infrastructure.

4.4.1 Functional perspective

Some studies are focused on a concrete subset of BDA applications, scientific workflow management systems. Almorsy and Grundy propose in *S1* [1] a visual DSL to describe sequential programs, planned parallelization aspects, and program deployment details in the domain of data-intensive scientific applications. Their proposal uses code generation of the corresponding parallel program using necessary parallel and distributed programming models (MPI, Open CL, or Open MP). Belli et al. in *S3* [11] present a cloud architecture for big stream applications based on standard protocols and open-source components, which provides an IoT processing platform. Their architecture includes the acquisition, normalization, application register, and graph-network modules composed by listeners and designed to be open and extensible. Kashlev and Lu in *S16* in [54] identifies the key challenges for running scientific big data workflows in the cloud and proposes DATAVIEW, an implementation-independent system architecture. DATAVIEW allows users to design, configure, and deploy workflows over multiple cloud infrastructures.

Another group aims at modeling approaches to specifying ML or data mining applications from high-level abstractions. Lechevalier et al. in *S21* [63] introduce a DSM framework for predictive analytics of manufacturing data using artificial neural networks to generate analyt-

Table 4.4.1: Primary Studies

ID	Author(s)	Type	Year
S1	Almorsy and Grundy[1]	Workshop	2015
S2	Anandan et al.[4]	Conference	2015
S3	Belli et al.[11]	Workshop	2015
S4	Bersani et al. [13]	Workshop	2016
S5	Camillieri et al. [15]	Workshop	2016
S6	Chen et al.[27]	Conference	2016
S7	Chen et al.[25]	Workshop	2015
S8	Eichelberger et al.[34]	Conference	2016
S9	Gil et al.[40]	Conference	2017
S10	Gómez et al. [41]	Workshop	2016
S11	Gorton and Klein[42]	Journal	2015
S12	Guerriero et al.[46]	Workshop	2016
S13	Harper et al.[47]	Conference	2015
S14	Huang et al.[50]	Conference	2015
S15	Jovanovic et al.[53]	Workshop	2012
S16	Kashlev and Lu[54]	Conference	2014
S17	Tuovinen et al.[95]	Conference	2008
S18	Openscoring [74]	N/A	2020
S19	FastScore [73]	N/A	2020
S20	Gribaudo et al.[44]	Journal	2017
S21	Lechevalier et al.[63]	Conference	2015
S22	Sujeeth et al.[91]	Conference	2011
S23	Artac et al.[6]	Conference	2018
S24	Perez-Palacin et al.[77]	Journal	2019
S25	Alrifai et al.[2]	Tech Report	2014

ics models. Sujeeth et al. present in *S22* [91] OptiML, a DSL for machine learning which describes analytics functions using a statistical model which covers a subset of ML algorithms, these analytics functions are analyzed and optimized before the code generation. Jovanovic et al. in *S15* [53] propose a tool to produce multi-dimensional and ETL (data extraction, transform, and load) conceptual designs semi-automatically from service-level agreements and data source descriptions formally expressed. Then, this tool translates the semantic conceptual designs into physical designs to be deployed on a database management system and ETL engines. Their proposal checks how multi-dimensional concepts map to the sources and detects model ambiguities.

QualiMaster *S8, S25* [2] focuses on the processing of online data streams for real-time applications, such as the risk analysis of financial markets regarding metrics of time behavior and resource utilization. QualiMaster aims to maximize the throughput of a given processing pipeline. Eichelberger et al. in [34] extends Qualimaster to supports topology modeling based on configuration constraints and variability refinements. These capabilities are part of the general-purpose Integrated Variability Modeling Language (IVML). A BDA application is modeled using IVML, specifying the pipeline structure, analytics algorithm family, data sources, and infrastructure. In the same line of variability models, Camillieri et al. in *S5* [15] introduce the ROCKFlows project, a software platform to support the construction of ML workflows from the software product line approach. ROCKFlows incorporates feature models to specify ML algorithms, define a concrete configuration, and code generation to implement ML workflows.

4.4.2 Infrastructure perspective

In contrast, we found another group of studies interested in architecture and infrastructure concerns of BDA applications leaving aside their functional components.

Gribaudo et al. in *S20* [44] propose a modeling framework based on graph-based language to evaluate the system's performance of running applications that follow the lambda architecture pattern. This modeling framework allows users to define stream, batch, storage, and computation nodes and performance indices to be simulated and evaluated. Huang et al. in *S14* [50] introduce a model to design, deploy, and configure Hadoop clusters through architecture metamodel and rules, which describe BDA infrastructure and deploy automation. Their work is focused on the design, deployment, and evaluation of BDA technology infrastructures. Tuovinen et al. in *S17* [95] present an extension with a new layer of execution logic of pre-existing framework to develop data mining application using a pipe and filter architecture. In addition, this architecture provides parallel distributed processing to support data-intensive applications

composed of a database, an input receiver, and a user interface back-end.

A group of studies come from the software architecture field and proposes architecture methodologies, references, tactics, or patterns. Chen et al. in *S6* and *S7* [27] [25] analyze BDA projects to identify deployment challenges and the importance of architecture design choices and technology selection to tackle them. In addition, they address these challenges by adopting an architecture-centric methodology: ABBA (Architecture-centric Agile Big data Analytics). Gorton and Klein in *S11* [42] propose an extension of architecture tactics to support big data systems. In addition, they also link tactics to design solutions based on specific big data technologies, enabling architects to rapidly relate the capabilities of a particular technology to a specific set of architecture tactics. Harper et al. *S13* [47] introduce an architectural pattern for the design of flexible BDA platforms, which is extensible using different pipelines. This pipeline pattern aims to accelerate the implementation of industrial analytics applications driven by quality attribute trade-offs.

Openscoring (*S18*) [74] is an open-source software suite that supports predictive model evaluation, conversion, and deployment based on PMML compatibility. The Openscoring service can publish, upgrade, and consult PMML models as API REST micro-services to be consumed by clients. This suite promotes interoperability and exploits neutral-technology models for the analytics scoring capability, but deployment strategies, monitoring, ingestion, and sink operators are beyond the scope. Openscoring is built on the JPMML library, which also provides model verification through dataset inputs and known results that can be used to validate the output accuracy regardless of the environment.

Fastscore in *S19* [73] is a commercial framework to design and deploy analytics models. Analytics components are conventionally developed using a determined programming language or a PMML file, and once imported to the platform, they can be connected to data inputs and outputs. Quality scenarios cannot be specified, but performance metrics can be visualized. Deployment is realized through *engines* (containers) where models are executed, and the deployment design is limited to engine replication to increase the concurrency.

SpringXD (now called Spring Cloud Data Flow) in *S2* [4] is a unified, distributed, and extensible system for data ingestion, analytics, processing, and export to simplify BDA development and deployment. In SpringXD, modules are data processing units of one of three types: *source*, *processor*, or *sink*, and they can be connected using messaging abstractions called *message bus* to build BDA pipelines. Modules run over a cluster of containers which can be replicated to a fixed number and monitored to observe performance behavior, although these metrics are not application-oriented but infrastructure-oriented (e.g., CPU and memory use). Similar to our

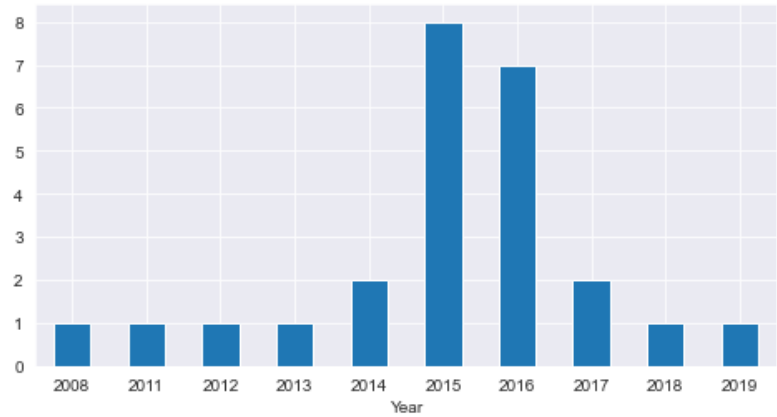

Figure 4.5.1: Studies published per year.

approach, analytics processors can be defined through PMML models, but target technologies are limited to a set of predefined options.

DICE project in *S4, S9, S10, S12, S23, and S24* [13][40][41][46][6][77] presents a DSM offering big data design which comprises data, computation, technology-frameworks, and deployment concepts to design and deploy data-intensive applications. DICE proposes a model-driven engineering approach to develop application models which are automatically transformed into IaC. In addition, DICE includes quality of service requirements associated with elements within the application, which are analogous to QS. Perez et al. in [77] present a profile to enable performance and reliability assessment in DICE. DICE supports configuration management, service provisioning, and application deployment, but technology-neutral models and architectural tactics are not considered, hindering portability and design decision tracing. Due to its focus, DICE requires design at a very detailed level, specifying different constructs regarding target technologies. However, in our approach, the technology-specific generators transform functional and deployment artifacts to code.

4.5 Discussion

Figure 4.5.1 details publication years, and it shows growth from 2014 to 2017. The Figure 4.5.2 reports the frequency of studies per publication type. Most of the studies are conferences (11) and workshops (8), followed by three journals and two industry products which have not yet published studies. Figure 4.5.3 depicts the frequency of studies per context type (Industry/A-

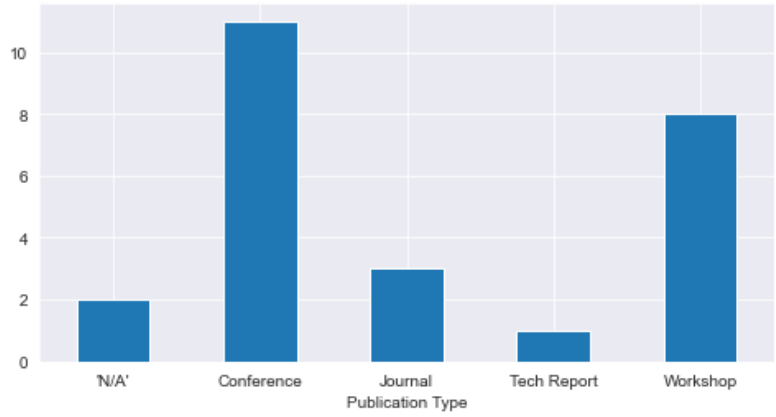

Figure 4.5.2: Studies per publication type.

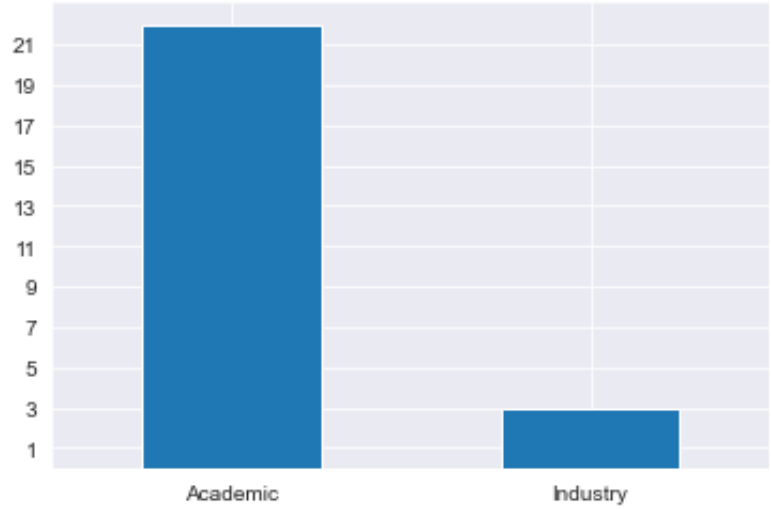

Figure 4.5.3: Studies per context type.

cademy), where 22 studies are developed in the academy and 3 in industry.

4.5.1 Quality evaluation

We evaluated the studies using quality criteria detailed in Table 4.3.3. We grouped studies that are part of the same work/project. In the case of studies S4, S9, S10, S12, S3, and S24, they report incremental deliveries of *DICE project* [33]. Also, studies S8 and S25 were within the Qualimaster project. As well as studies S6 and S7 [27][25] that present proposal and validation of *Architecture-centric Agile Big data Analytics (AABA)*. The scoring results for each criterion and study are presented in Tables 4.5.1 and 4.5.2.

Table 4.5.1: Quality Evaluation Results from QC1 to QC5.

Id	Product	QC1	QC2	QC3	QC4	QC5					
						Bus.	Ing.	Pre.	Mod.	Depl.	Oper.
S1	F	3.0	Y	Y	Y	0.0	0.0	3.0	3.0	4.0	0.0
S2	F	4.0	Y	N	Y	0.0	5.0	4.0	4.0	5.0	4.0
S3	F	4.0	N	N	Y	0.0	5.0	4.0	2.0	1.0	0.0
S4,S9,S10,S12,S23,S24	F	5.0	Y	N	Y	0.0	4.0	4.0	2.0	5.0	5.0
S5	F	3.0	Y	Y	Y	2.0	2.0	3.0	4.0	3.0	2.0
S6,S7	M	5.0	N	Y	Y	5.0	1.0	1.0	1.0	1.0	0.0
S8,S25	F	2.0	Y	N	N	0.0	2.0	2.0	3.0	4.0	4.0
S11	C	2.0	N	Y	Y	2.0	0.0	0.0	1.0	3.0	1.0
S13	C	4.0	N	Y	Y	0.0	1.0	1.0	1.0	0.0	0.0
S14	F	1.0	N	N	Y	0.0	0.0	0.0	0.0	4.0	1.0
S15	F	1.0	N	N	Y	4.0	3.0	4.0	0.0	0.0	0.0
S16	F	1.0	N	N	Y	0.0	3.0	3.0	3.0	4.0	2.0
S17	F	3.0	N	N	Y	0.0	3.0	3.0	3.0	3.0	2.0
S18	F	4.0	Y	N	Y	0.0	1.0	3.0	4.0	2.0	3.0
S19	F	4.0	Y	Y	Y	0.0	4.0	4.0	4.0	4.0	4.0
S20	F	3.0	N	Y	Y	0.0	1.0	0.0	0.0	4.0	4.0
S21	F	1.0	Y	Y	N	2.0	0.0	0.0	4.0	0.0	0.0
S22	F	3.0	N	Y	Y	0.0	0.0	4.0	4.0	4.0	0.0

Table 4.5.1 reports the product generated (i.e., F: Framework, M: Methodology or C: Catalog), quality criteria related to the type of validation (QC1), separation of concerns (QC2), technology–neutrality (QC3), cross-industry support (QC4), and coverage of analytics life cycle (QC5). The best QC1 scoring (5.0) is assigned to *DICE project* (S4, S9, S10, S12, S23, and S24) and S6-S7 studies. In terms of QC2, QC3, and QC4, studies S1, S5 and S19 meet all criteria. Finally, to obtain an overall score for QC5, it is necessary to average the sub-criteria: business goals, ingestion, pre-processing, modeling, deployment, and operation. Thus, the best ratings for QC5 are achieved by *S2-SpringXD* (3.7), *DICE* (3.3), and *S19-Fastscore* (3.3).

Table 4.5.2 provides the quality evaluation of the remaining criteria related to software architecture practice constituted by QC6 and QC7 subcriteria. The overall scoring is worse than the functional criteria. By averaging each sub-criteria, the best scoring were achieved by *DICE* (3.3), *Qualimaster-S8,S25* (2.8), and *SpringXD-S2* (2.3). It is worth noting that most studies do not address tactics and patterns in BDA, neither which present a niche to occupy. Also, the validation or verification of BDA architectures is rarely addressed (only two studies), which can adversely impact the deployment when some crucial constraints are not guaranteed.

Table 4.5.2: Quality Evaluation Results of QC6.

Id			QC6			QC7
	QS	Tactics / Patterns	DevOps	Monitoring	Code Gen.	Validation
S1	0.0	0.0	1.0	0.0	Y	N
S2	0.0	0.0	5.0	4.0	Y	N
S3	0.0	0.0	0.0	0.0	N	N
S4, S9, S10, S12, S23, S24	4.0	0.0	5.0	4.0	Y	Y
S5	3.0	0.0	1.0	1.0	N	N
S6, S7	4.0	3.0	1.0	0.0	N	N
S8, S25	4.0	0.0	3.0	4.0	Y	Y
S11	3.0	3.0	0.0	0.0	N	N
S13	3.0	2.0	0.0	0.0	N	N
S14	1.0	0.0	4.0	2.0	Y	N
S15	3.0	0.0	2.0	0.0	Y	Y
S16	1.0	0.0	3.0	1.0	N	N
S17	0.0	2.0	0.0	0.0	N	N
S18	0.0	0.0	2.0	2.0	N	Y
S19	0.0	0.0	4.0	4.0	N	N
S20	3.0	0.0	2.0	4.0	N	Y
S21	0.0	0.0	0.0	0.0	Y	N
S22	0.0	0.0	0.0	0.0	Y	N

4.6 Summary

The results obtained demonstrate a growing interest and work in BDA deployment, mainly in terms of the coverage of analytics phases, but there are under-explored fields such as separation of concerns, QS, business objectives, tactics, patterns, and DevOps.

The best overall ranked approaches based on the quality criteria were *DICE* (S4, S9, S10, S12, S23, S24), *SpringXD* (S2), *Fastscore* (S19), and *Qualimaster* (S8, S25). DICE[1] is a quality-aware DevOps for Big Data applications supported by a research collaboration among nine European organizations and sponsored by the Horizon 2020 program. DICE offers a model-driven approach to build analytics pipelines regarding QS expressed using the MARTE UML profile. SpringXD, now called Spring Cloud Data Flow[2], is a Spring open-source project based on micro-services architecture to implement streaming and batch data processing for Cloud Foundry and Kubernetes. Spring Cloud Data Flow provides tools to create complex workflows ranging from ETL to import/export, event streaming, and predictive analytics. Qualimaster[3] is a configurable real-time data processing infrastructure that enables the construction of financial risk scoring ap-

[1] http://www.dice-h2020.eu/
[2] https://spring.io/projects/spring-cloud-dataflow
[3] http://qualimaster.eu

plications. Fast score [4] (now called ModelOp) is a commercial software solution to deploy analytic models and to monitor their performance that allows configuring sources, data schemes, and analytics models.

These approaches offer a set of functions and DevOps capabilities, but they do not offer an integrated viewpoint for business users, data scientists, and IT architects. Instead, they only focus on IT developers with analytics backgrounds where separation of concerns is not addressed. Additionally, they are constrained to specific technologies, and there is not support to select the technology that best fits business and quality requirements. Finally, these studies fall short of validating or verifying BDA architecture and applications before the deployment process. This review enabled us to identify potential unexplored fields in BDA deployment from a software architecture perspective using DSM and DevOps practices.

[4] https://www.modelop.com/blog/open-data-group-announces-fastscore-release-1-10-2

Part II

ACCORDANT

5

ACCORDANT Overview

Part I described the state-of-the-art and the state of the practice in BDA development and deployment. In this chapter[1], we present ACCORDANT (An exeCutable arChitecture mOdel foR big Data ANalyTics) to bridge the gap of current approaches found in Part I. This chapter presents an ACCORDANT overview, its components, and how they are related.

5.1 The ACCORDANT approach

From the first low-level machine code to today's high-level programming languages which are closer to the problem domain, raising the levels of abstraction has been a crucial part of software

[1]Portions of this chapter previously appeared in: Castellanos, C., Varela, C. A., Correal, D. (2021). "ACCORDANT: A Domain Specific-Model and DevOps Approach for Big Data Analytics Architectures". Journal of Systems and Software, 172, 110869. C. Castellanos, B. Perez, D. Correal and C. Varela, "A Model-Driven Architectural Design Method for Big Data Analytics Applications". In 2020 IEEE International Conference on Software Architecture Companion (ICSA-C), Salvador, Brazil, 2020 pp. 89-94. Castellanos, C., Varela, C. A., Correal, D. (2019, September). "Measuring performance quality scenarios in big data analytics applications: a DevOps and domain-specific model approach". In Proceedings of the 13th European Conference on Software Architecture-Volume 2 (pp. 165-172). Castellanos C., Correal D., Rodriguez JD. (2018) "Executing Architectural Models for Big Data Analytics". In: Cuesta C., Garlan D., Pérez J. (eds) Software Architecture. ECSA 2018. Lecture Notes in Computer Science, vol 11048. Springer, Cham.

engineering. Domain-specific modeling (DSM) is proposed to aim at this goal with the notion that models should be primary artifacts in software development instead of merely supporting documentation. DSM languages support domain experts in working within their vocabulary and problem space without being concerned about technical details of the solution space (e.g., infrastructure, programming languages, and middleware). DSM also provides an accessible way to communicate with stakeholders who are not familiar with the fast-changing technologies, as stated in Part I.

In addition, a DSM approach enables us to specify restrictions and detect architectural mismatches on BDA by formalizing domain-specific constraints. This formalization makes explicit the architecture assumptions and allows us to validate them automatically. On the other hand, reference architectures (RA) aim at guiding the implementation of software architectures within a particular domain such as BDA. An RA allows the definition of domain-specific components and provides a common vocabulary to discuss different implementations. Hence, an RA helps guide software design processes and make them efficient, effective, and interoperable [5, 65].

Our strategy takes into account the previous arguments to introduce a DSM approach to address the problems stated in Part I. We aim to reduce the deployment time, improve architectural design and analysis, and enable architects to check mismatches in BDA architectures. Our strategy is to raise the level of abstraction of BDA architecture specifications using four main components: 1) a BDA reference architecture to guide the architecture design; 2) an architecture modeling framework to formally specify and integrate multiple views and artifacts from different stakeholder domains; 3) an architecture validator to check constraints over concrete architecture models; and 4) a method to design, develop, deploy, and operate a BDA application integrating the previous components into a sequence of activities. Fig. 5.1.1 provides an ACCORDANT conceptual model which describes its components (boxes), and how they are related (arrows).

- The ACCORDANT reference architecture offers guidance by providing functional, integration, and technology views along with a catalog of tactics and patterns for BDA. Besides, this reference architecture works as a conceptual framework to implement the concrete modeling framework.

- The ACCORDANT Modeling Framework (MF) is a DSM approach that uses the ACCORDANT Metamodel and bases its building blocks (patterns, tactics, components) on the reference architecture. ACCORDANT MF allows architects to design and deploy architecture models using domain-specific languages. This modeling framework separates

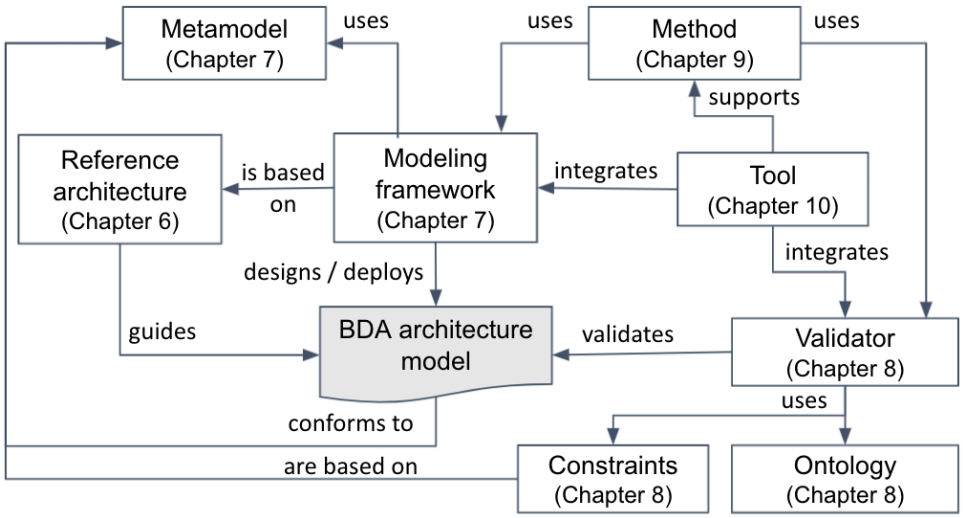

Figure 5.1.1: ACCORDANT overview

the BDA domains enabling stakeholders to specify their part of the problem in their vocabulary: business, data science, and IT architecture.

- The ACCORDANT Validator uses a set of constraints and an ontology to check architectural properties over architectural models which are conformed to the ACCORDANT metamodel.
- The ACCORDANT Method defines a set of activities to design, validate, deploy, and operate BDA applications. ACCORDANT MF and the architecture validator implement the ACCORDANT Method activities.
- The ACCORDANT Tool supports the ACCORDANT method integrating both the ACCORDANT MF and Validator.

5.2 ACCORDANT Method overview

This section presents a general description of the ACCORDANT Method to understand the sequence of activities that we present to design, deploy, and operate a BDA application. This method overview also facilitates the presentation of each ACCORDANT component in an ordered sequence. The ACCORDANT Method is cyclic, iterative, and composed of seven activities across four software development life-cycle phases as depicted in Fig. 5.2.1: Requirements elicitation, development, deployment, and operation. The activities in the ACCOR-

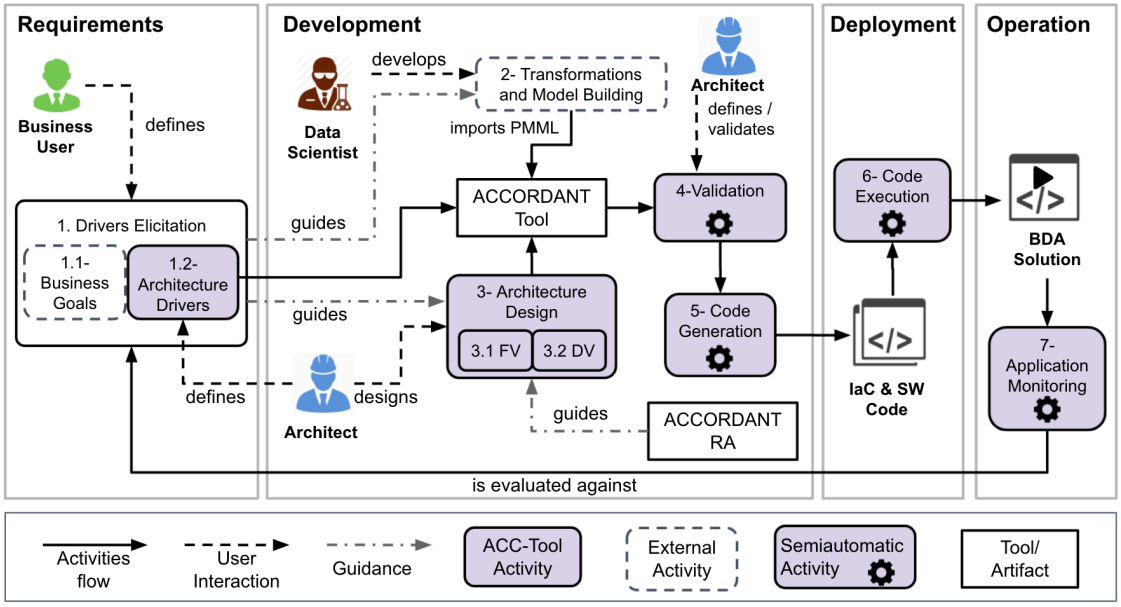

Figure 5.2.1: ACCORDANT Method Overview

DANT Method can be composed of tasks. The activities and tasks framed in solid lines are performed using the ACCORDANT MF directly, while activities and tasks made with external tools are framed in dotted lines. The activities marked with a gear icon are executed semi-automatically by ACCORDANT (see activities 4, 5, 6, and 7). We will revisit this method in Chapter 9 where its usage with the ACCORDANT artifacts will be detailed using a running example.

The business user defines 1.1) business goals and 1.2) architecture drivers, which will guide the following activities. 2) The data scientist develops data transformations, builds and evaluates analytics models. The resulting analytics models are exported as PMML files. 3) The architect designs the software architecture using the ACCORDANT metamodel in terms of *Functional Viewpoint*(FV) and *Deployment Viewpoint*(DV). FV model makes use of PMML models to specify the software behavior. 4) FV and DV models are validated against architecture constraints. 5) Code generation of software and infrastructure is performed from integrated models. 6) The code generated is executed to provision infrastructure and install software artifacts. 7) The application behavior is monitored in operation to be evaluated against QS as defined in the architecture drivers. New method iterations can be triggered when adjustments or upgrades are required. The following chapter will offer a deeper explanation of ACCORDANT's components introduced in this chapter.

6

A reference architecture (RA) for big data analytics

This chapter[1] presents an RA for BDA to address the challenges found in Chapters 3 and 4 related to lack of software architecture perspective, design guidance, and technology selection. This RA includes architectural patterns, tactics, and technology catalogs to guide Activity 3 in the ACCORDANT Method (architecture design) and to address Research Objective *RO3*.

6.1 Introduction

We have highlighted challenges in the previous chapters' BDA development and deployment process, procedures, techniques, and tools. Software architecture practices support the development and deployment of BDA applications to design systems driven by QS, tactics, patterns, and guides. In addition, the extensive and growing BDA technologies along with multiple big data sources and diverse analytical techniques provide a large number of options to implement BDA

[1]Portions of this chapter previously appeared in: Castellanos C., Perez B., Correal D. (2021) "Smart Transportation: A Reference Architecture for Big Data Analytics". In: Khan M.A., Algarni F., Quasim M.T. (eds) Smart Cities: A Data Analytics Perspective. Lecture Notes in Intelligent Transportation and Infrastructure. Springer, Cham.

applications [39]. This large number of options delays the analysis, selection, and integration of software components that architects must complete. Previous industry surveys [17, 26, 82] have shown challenges on BDA applications deployment, including delayed time-to-market, interoperability, and immature deployment procedures, and technology heterogeneity. The complexity to design and develop BDA projects requires a set of software architecture guidelines and catalogs to facilitate and accelerate the adoption of these cutting-edge technologies.

Reference architectures (RA) guide the implementation of software architectures within a particular domain, such as BDA. Chen et al. [25] evidenced that the use of reference architectures and technology catalogs support BDA design and development in practice. An RA facilitates the definition of good practices, provides a common vocabulary on which to discuss different implementation proposals, and accelerates the deployments by reusing effective solutions. An RA could be technology-neutral but also can include a technology assignment model or technology blueprint to guide software design processes and make them efficient, effective, and interoperable [5, 65].

Although previous works have proposed RAs for BDA [23, 39, 72, 80, 86, 101], these works are focused on the functional or in technology components in a separated facet. Nevertheless, they do not provide an integrated RA which aligns technology-neutral services to catalogs of architectural tactics, patterns, and technologies. To tackle these issues, we present an RA for BDA [20] which addresses the challenges and needs described above. The contributions of this chapter are summarized as follows:

- A review of software and reference architectures for BDA.
- A reference architecture for BDA.
- A catalog of BDA patterns and tactics.

The rest of the chapter is organized as follows: Section 6.2 presents the RA for BDA. Section 6.3 summarize patterns and tactics used in this domain. Finally, Section 6.4 summarizes the chapter.

6.2 A reference architecture for big data analytics

This section presents ACCRA (ACCORDANT RA), an RA to guide the planning, development, and deployment of applications in this domain. According to Angelov et al. [5], our RA is classified as a *Type 3*, a classic RA aiming to facilitate the implementation of BDA solutions for multiple organizations proposed by an independent research center. ACCRA comprises detailed

views and catalogs, including the definition of candidate technologies and integration protocols. ACCRA also includes the stages of data acquisition, storage, extraction, processing, and exploitation.

RAs are described by views, each one representing specific systems concerns. These views are of particular interest to a stakeholder and facilitate the communication, and discussion [9]. In particular, the views included in ACCRA are functional, technology, and integration. The functional view presents services (set of common features), connectors (communication between services), and service groups, see Fig. 6.2.1. The *Presentation Zone* (A) defines how the services provide information through channels. The zone (B) details the *Backend Services*, which gathers the domain-specific services and integration services.

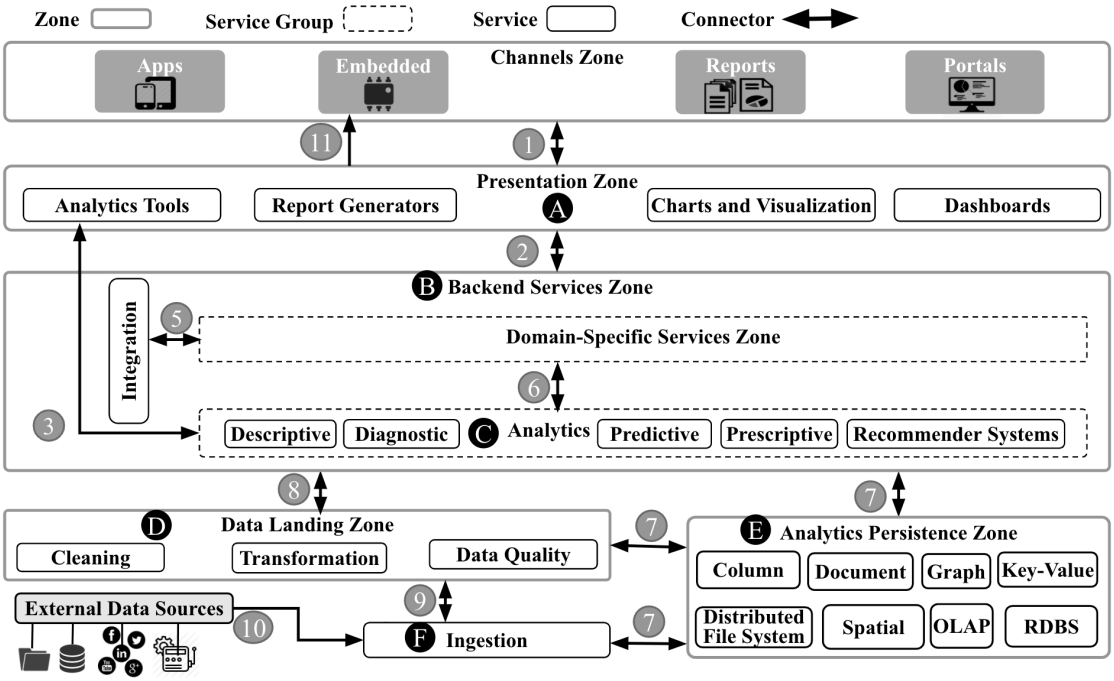

Figure 6.2.1: ACCRA - Functional View

The *Analytics Zone* (C) indicates the types of analysis that can be performed, which are supported by the services exposed in the *Data Landing Zone* (D) and *Analytical Persistence Zone* (E). *Analytical Persistence Zone* offers services such as distributed file systems, SQL, and Non-SQL storage. Finally, the *Ingestion Zone* (F) has the responsibility of collecting external data sources and redirecting them to the corresponding consumers.

The integration view is based on the Trivadis Integration Architecture Blueprint [87] to describe each type of connector (numbered from *1* to *11* in Fig. 6.2.1) of the transport, distribu-

tion, mediation, and application layers. Fig. 6.2.2 details connector type 10, describing the integration between external data and the ingest service. This connector details the protocols recommended to integrate external sources in the integration and transport layer, such as HTTP, FTP, WebSocket, and MQTT. These protocols have to be selected regarding communication schema (connection-oriented or Pub/Sub) and their endpoints (e.g., web application, database, IoT device, FTP server).

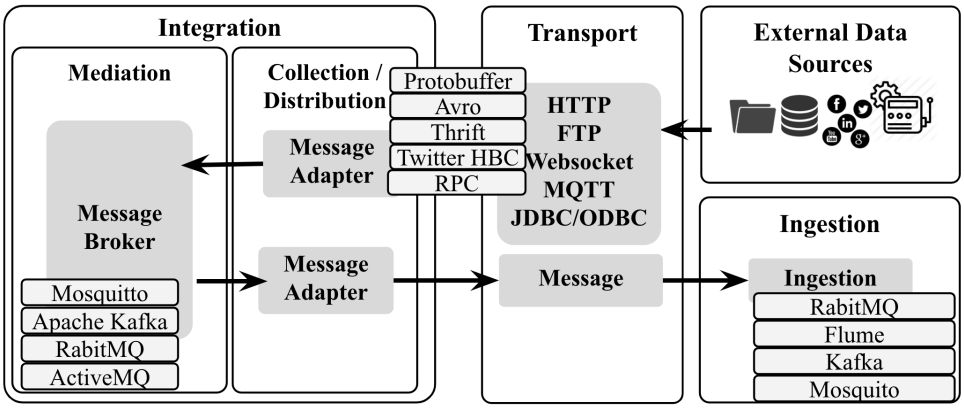

Figure 6.2.2: ACCRA - Integration View of Connector Type 10

The technology selection view provides a list of recommendations on specific products that offer the capabilities required by a service or group of services to be instantiated. This view works as a guide in the implementation of specific solution architectures. Fig. 6.2.3 details the technology view for open source technologies; other versions of this view can focus on specific vendor products.

6.3 Architectural tactics and patterns catalog for BDA

An RA guides the software architecture design to fulfill the system quality attributes, and this design is built on building blocks called patterns and tactics. Architectural patterns are well-established packaged strategies for solving some recurrent problems. Tactics are design decisions that influence the achievement of a quality attribute response such as performance, scalability, availability, modifiability, and security [9]. In this section, catalogs of patterns and tactics are compiled to support a systematic and guided architecture design. These catalogs are not intended to be comprehensive, but a representative collection of key and standard architectural

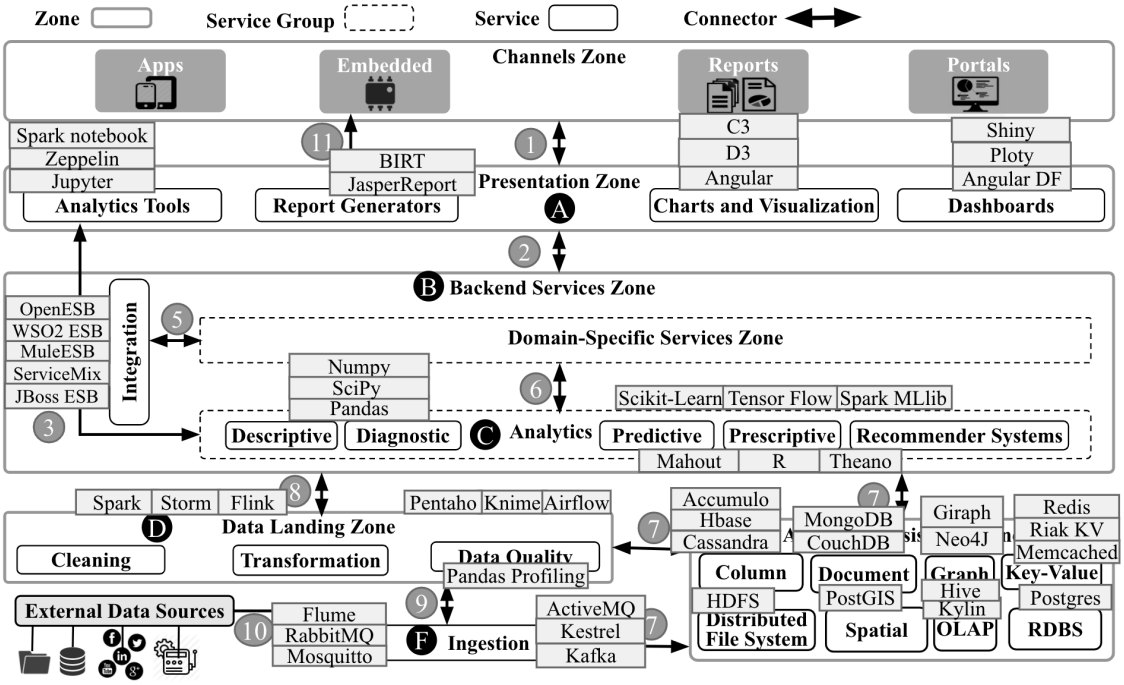

Figure 6.2.3: ACCRA - Technology Selection View for Open Source

building blocks found in the literature applicable in the BDA context.

6.3.1 A catalog of BDA tactics

Previous works have reviewed and proposed a set of tactics that are highly applied to BDA systems [9, 83], or designed specifically for big data cybersecurity [96] and big data databases [61]. Hence, in this section, we identify and classify these tactics in a unified catalog. Table 6.3.1 summarize tactics to achieve the quality attribute(s), and their source references. This catalog is a distilled and synthesized collection of available tactics for the specific BDA domain. We have not included tactics for security and modifiability deliberately because we consider them cross-cutting decisions for overall systems. Regarding these considerations, the collected tactics are described below, but more information can be found in the respective reference.

- *Increase available resources*: The scaling up or out can reduce latency by including faster processors, additional processors, additional memory, and faster networks [9].
- *Introduce concurrency*: Concurrency can be introduced to reduce blocked time by processing on different threads, increasing the overall throughput [9].

Table 6.3.1: Tactics for BDA

Tactic	Quality Attributes	References
Increase available resources	Performance	[9, 83]
Introduce concurrency	Performance	[9, 83]
Maintain multiple copies of data	Performance	[9, 96]
Maintain multiple copies of computations	Performance	[9, 83]
ML algorithm optimization	Performance	[96]
Unnecessary data removal	Performance	[96]
Feature selection and extraction	Performance	[96]
Parallel processing	Performance	[83, 96]
Reuse previous results	Performance	[83]
Result polling and optimized notification	Performance	[96]
Data cutoff	Performance	[96]
Shard data set across multiple servers	Performance	[61]
Asynchronous processing	Performance, Scalability	[83]
Load balance across replicas	Performance, Scalability	[61]
Relax transactional consistency	Performance, Scalability, Availability	[83]
Automatically rebalance data across nodes	Performance, Availability	[61]
Monitor data ingestion	Availability	[96]
Automatically maintain cluster membership list	Scalability, Availability	[61]
Elastic resources provision	Scalability	[61]
Combine multiple analytics models	Accuracy	[96]

- *Maintain multiple copies of data*: It involves keeping copies of data with different access speeds. Data replication implies keeping separate copies to reduce conflicts over concurrent accesses [9, 96].

- *Maintain multiple copies of computations*: Multiple processors to reduce the contention of computation compared to single server [9].

- *ML algorithm optimization*: Selecting the machine learning algorithm that is most efficient or tunes the algorithm to achieve a lower computational complexity [96].

- *Unnecessary data removal*: Remove data, records, or features that do not contribute to the analytics goal, e.g., duplicated data or data with low variance [96].

- *Feature selection and extraction*: Select and extract data characteristics that contribute more to explain the target variable [96].

- *Parallel processing*: Split extensive and lengthy processes into subprocesses, and consolidate them into a single result when the last subprocess is complete [83, 96].

- *Reuse previous results*: Reuse the results of the costly processing by caching and recycling them whenever needed in the following computations [83].

- *Result polling and optimized notification*: The mapper node notifies the reducer as soon as values change to a defined threshold, and accordingly, the mapper forwards the updated results [96].

- *Data cutoff*: Use data samples to reduce the size of the dataset, improving the overall system performance [96].

- *Shard data set across multiple servers*: Sharding improves the system's capacity by splitting the data set over a cluster, reducing the amount of data stored on each node, and improving performance by reducing I/O contention [61, 96].

- *Asynchronous processing*: Improve response times using asynchronous processing to reduce resource contention [61, 83].

- *Load balance across replicas*: Create multiple replicas of each object to improve read performance by allowing read requests to be balanced across multiple servers, and readers can access a closer copy, reducing read latency.

- *Relax transactional consistency*: The need for immediate system-wide consistency can be relaxed by allowing parts of the system to receive updates at slightly different times to be consistent eventually [83].

- *Automatically rebalance data across nodes*: Automatically reallocate the data across the nodes according to the number of available nodes, reassigning or resizing data shards [61].

- *Monitor Data ingestion*: Monitor the flow of streaming data to detect and block traffic peaks to prevent system crashes [96].

- *Automatically maintain cluster membership list*: A gossip protocol to manage the cluster membership list automatically that provides an eventual consistency of cluster across all nodes [61].

- *Elastic resources provision*: Allocate and deallocate nodes based on the computing demand enables infrastructure to fit dynamically with fluctuating workloads in a cost-effective way [61].

- *Combine multiple analytics models*: Integrate the results of multiple analytic models to reduces false positives and increase the accuracy [96].

6.3.2 A catalog of BDA architectural patterns

The difference between architectural style and patterns is still an open discussion. Similar to previous works [22, 29], this paper uses the term "Architectural Pattern" as a particular recurring design problem that arises in specific design contexts and presents a well-proven architectural generic scheme for its solution. Previous works [22, 62, 66, 69, 94] have proposed BDA patterns, and we have compiled them in this catalog summarized in Table 6.3.2, specifying the pattern and the problem to be addressed. The architectural patterns are described below, and information can be found in the respective reference.

- *Lambda Architecture*: Combine batch processing and stream processing applied to unbound datasets to obtain real-time views and aggregated views with higher accuracy in the long term. Hence, this pattern includes speed, batch, and serving layers [66].

- *Kappa Architecture*: This is a simplification of Lambda without the batch layer, reusing the speed layer to reprocess historical data with a more extended retention policy [62].

- *Poly Storage*: This pattern stores data with high volume, velocity, variety enabling availability for streaming, and random access [94].

- *Data Refinery*: This pattern cleans and structures non-relational data to load it in a relational data warehouse for further integration and analysis [94].

- *Data Lake*: A massively scalable repository that stores raw data centrally in their native format to be used by different processing/querying clients [69].

- *Interactive Query Engine*: This pattern executes distributed batch, and interactive analytic queries over high-volume data [22].

- *Distributed Message Broker*: Intermediary connector which offers high scalability by distributing messages along the cluster using a publish/subscribe approach [22].

- *Data Collector*: This pattern ingests, transforms, and delivers data for later use, supporting a wide variety of formats, velocities, and sources [22, 94].

- *Big Data Pipeline*: This is a compound pattern that comprises ingress, processing, and egress of big data integrating multiple sources and stages [94].

Table 6.3.2: Architectural Patterns for BDA

Pattern	Problem	References
Lambda	Combine stream and batch processing	[66]
Kappa	Combine stream and historical data processing	[62]
Poly Storage	Persist high-volume, high-velocity and high-variety data.	[94]
Data Refinery	Integrate relational and no-relational data sources	[22]
Data Lake	Store multi-structured raw data centrally	[69]
Interactive Query Engine	Ad hoc analytics queries over big data	[22]
Distributed Message Broker	Decouple producer and consumers with high availability and reliability	[22]
Data Collector / Ingestor	Collect high volume, speed, variety datasets from different sources	[22, 94]
Data Sink	Export high volume, speed, variety datasets to different sources	[94]
Analytics Processing Engine	Use a distributed parallel programming framework to pre-process or apply machine learning on large amounts of data	[22, 94]
Big Data Pipeline	Break out complex processing operations	[94]

6.4 Summary

In this chapter, we have presented ACCRA, an RA, to guide the design and deployment of BDA applications. ACCRA guided the definition and structuring of services in this domain and their integration. This research also supports the design of the DSM framework for BDA applications by providing technology, tactics, and patterns catalog, which will be the basis for the ACCORDANT MF in Chapter 7. ACCRA aims to facilitate the design and development of BDA architectures by offering patterns, tactics, and technology catalogs.

7

The ACCORDANT Modeling Framework (MF)

In chapter 6, we presented and classified tactics, patterns, and technology catalogs to guide the development of BDA applications. This chapter [1] presents the ACCORDANT MF to support the design, deployment, and operation of BDA solutions. Besides, ACCORDANT incorporates the architectural concepts defined in Chapter 6 to guide the development and deployment of BDA applications. ACCORDANT MF supports the ACCORDANT Method in task 1.2, and activities 3, 5, and 6. This chapter addresses the research objectives *RO4*, *RO5*, and *RO6*.

[1] Portions of this chapter previously appeared in: Castellanos, C., Varela, C. A., Correal, D. (2021). "ACCORDANT: A Domain Specific-Model and DevOps Approach for Big Data Analytics Architectures". Journal of Systems and Software, 172, 110869. C. Castellanos, B. Perez, D. Correal and C. Varela, "A Model-Driven Architectural Design Method for Big Data Analytics Applications". In 2020 IEEE International Conference on Software Architecture Companion (ICSA-C), Salvador, Brazil, 2020 pp. 89-94. Castellanos, C., Varela, C. A., Correal, D. (2019, September). "Measuring performance quality scenarios in big data analytics applications: a DevOps and domain-specific model approach". In Proceedings of the 13th European Conference on Software Architecture-Volume 2 (pp. 165-172). Castellanos C., Correal D., Rodriguez JD. (2018) "Executing Architectural Models for Big Data Analytics". In: Cuesta C., Garlan D., Pérez J. (eds) Software Architecture. ECSA 2018. Lecture Notes in Computer Science, vol 11048. Springer, Cham.

7.1 Introduction

ACCORDANT is a domain-specific model (DSM) approach to develop, deploy, and monitor BDA solutions bridging the gap between analytics and IT domains. ACCORDANT incorporates tactics, patterns, and technology catalogs presented in Chapter 6 to guide the development and deployment of BDA applications. In addition, ACCORDANT supports technology-neutral analytics models (PMML) to deploy them using DevOps and IaC mechanisms, promoting interoperability and speeding up the deployment process.

ACCORDANT allows us to design BDA applications using quality scenarios (QS), functional, and deployment views. A QS specifies a quality attribute requirement for a software artifact to support the design and quality assessment. The functional view defines the architectural elements that deliver the application's functionality. Deployment view describes how software is assigned to hardware-processing and communication elements. Our deployment strategy incorporates Kubernetes since containers offer consistent modularity to facilitate portability, continuous integration, and delivery. ACCORDANT offers an extension for traditional DevOps practice by starting from architectural artifacts instead of source code for operational deployment. We call this approach: "ArchOps" because we believe that architectural models are first-class entities in software development, deployment, and operation.

In summary, the specific contributions of this chapter are as follows:

- A DSM framework to formalize and accelerate iteratively developing and deploying BDA solutions by specifying architectural, functional, and deployment views aligned to QS.
- Three integrated meta-models specify architectural drivers, component-connector models, and deployments, thus accelerating the BDA deployment cycle.
- A containerization approach to promote automation delivery and performance metrics monitoring for BDA applications aligned to QS.
- A code generation process to produce functional and infrastructure code from ACCORDANT models.

The rest of this chapter is organized as follows. Section 7.2 details the ACCORDANT meta-models. Section 7.3 describes the code generation. Finally, Section 7.4 summarizes this chapter.

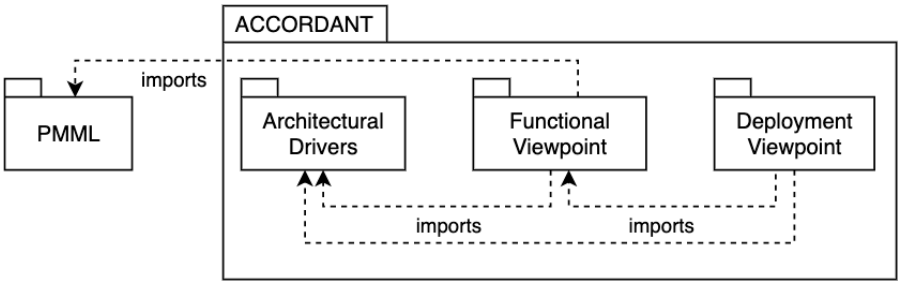

Figure 7.2.1: ACCORDANT's metamodel packages overview

7.2 ACCORDANT METAMODELS

The ACCORDANT metamodel includes the architectural drivers, *functional viewpoint* (*FV*) and *deployment viewpoint* (*DV*). Fig. 7.2.1 offers a high-level view of ACCORDANT metamodel packages which are developed as Ecore metamodels over the Eclipse modeling framework[2]. Architectural drivers are imported in the FV and DV to express how architectural decisions are implemented in components, connectors, and deployment constructs. The PMML metamodel is also imported into the FV to map PMML elements such as algorithms, data mining schema, and data types. The DV imports FV to specify how components and connectors are deployed. The following sections will detail each package and its content.

7.2.1 ARCHITECTURAL DRIVERS

According to architecture design methods such as Attribute-Driven Design (ADD)[102], the design is driven by predefined quality scenarios (QS), which must be achieved through design decisions compiled in well-known catalogs of architectural patterns and tactics. Both QS and tactics are drivers of the architecture design. Therefore business users and IT architects include these initial building blocks in the ACCORDANT metamodel and other concepts defined in ADD in the ACCORDANT Task 1.2. Fig. 7.2.2 details the main input building blocks grouped by the architectural project (*Project*) which contains the elements required by an architecture design: architecture constraints (*Constraints*), quality scenarios (*QScenario*), analyzed QS (*AnalyzedQS*), architectural decisions *ArchDecision*, and tactics (*Tactic*).

Constraints are properties about a system's design that must be incorporated into any final design of the system[102]. A *Constraint* has a code, a *ConstraintType*, and a value for this type

[2]https://www.eclipse.org/modeling/emf/

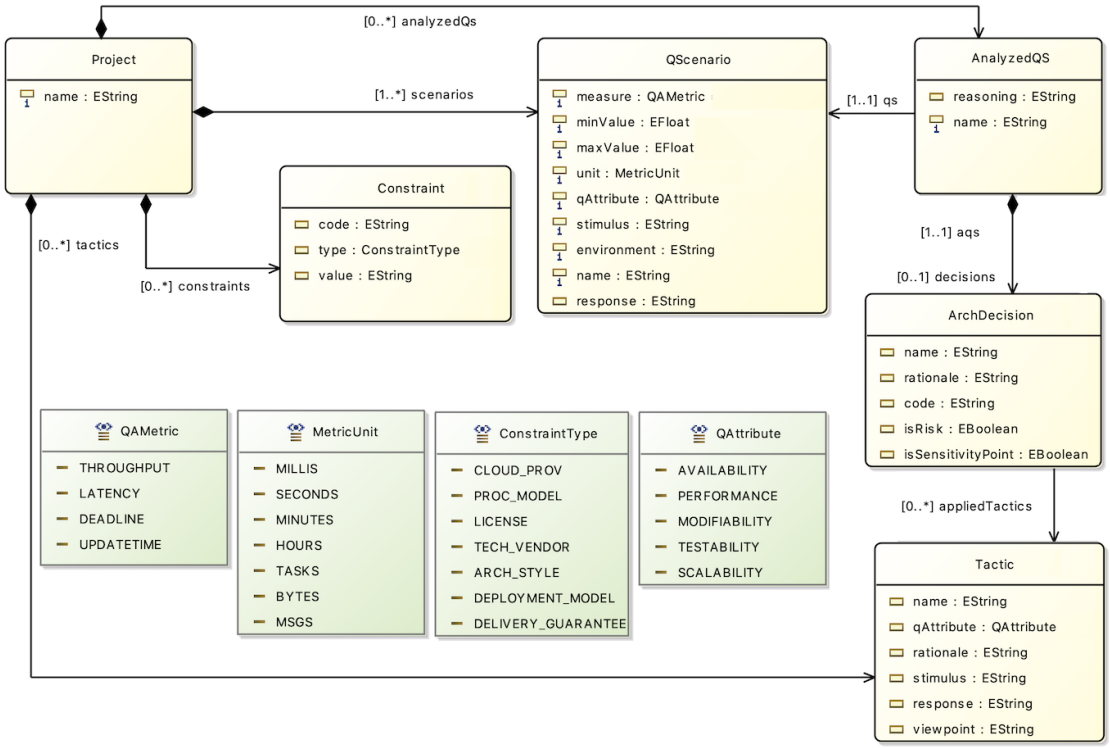

Figure 7.2.2: Excerpt of architectural drivers metamodel.

that has to be hold by the architecture. A *QScenario* determines a quality attribute requirement (*QAttribute*: i.e., availability, performance, scalability, etc.) for a system artifact, measured using a specific metric (*QAMetric*: i.e., latency, throughput, etc.). Thus, for instance, a QScenario could be defined as "latency $<= 3$ seconds for an artifact X", where artifact X corresponds to a software component or connector. IT architects analyze QS through an *AnalyzedQS* giving a rationale about how to achieve this QS through architectural decisions.

An *ArchDecision* is the result of the QS analysis and addresses the achievement of QS. Tactics and patterns catalogs guide these decisions, and they can affect different QAs, which implies a trade-off between competing QAs. For instance, an architectural decision that promotes availability by using multiple replicas can affect the performance since some synchronization must be done. If the architectural decision can be potentially problematic is recorded as *Risk*. Finally, a decision is a sensitivity point when it is critical for achieving a particular quality attribute response. Finally, the *Tactics* synthesize the BDA tactics catalog to be applied in an architecture instance, e.g., dynamic resource allocation, health monitoring, parallel processing, feature selection, etc. An architectural decision instantiates tactics in a concrete application context.

7.2.2 Functional Viewpoint (FV)

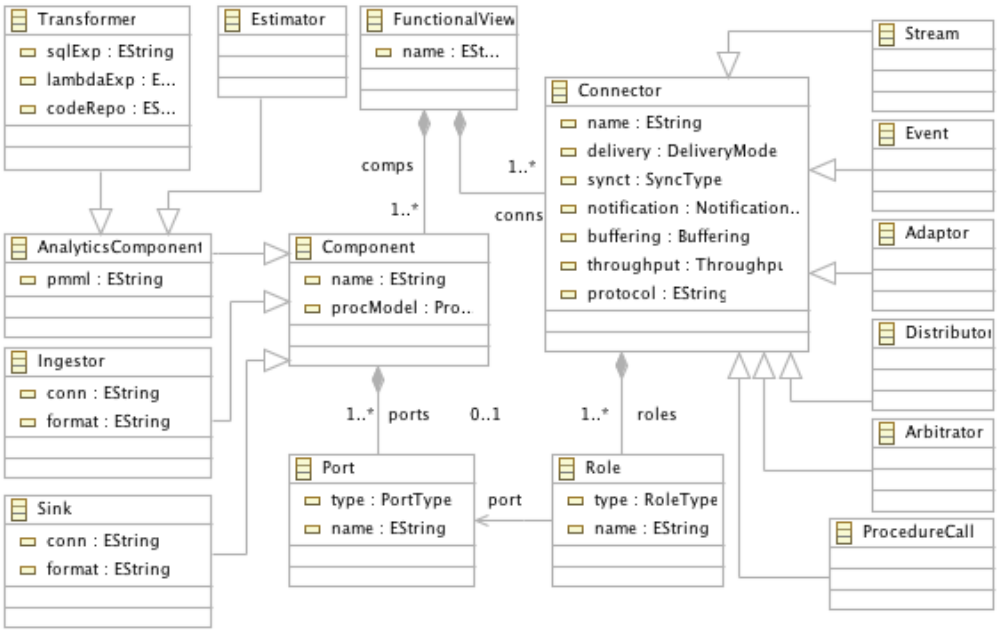

Figure 7.2.3: Excerpt of the Functional viewpoint metamodel

FV allows IT architect to design analytics pipelines in terms of ingestion, preparation, analysis, and sink building blocks in the ACCORDANT Task 3.1. *FV* specifies functional requirements for a BDA solution, and the constructs are described in a technology-neutral way as detailed in the metamodel depicted in Fig. 7.2.3. FV is expressed in a component-connector model. The *Component* represents an specific function or capability provided by the system, and they are specialized in *Ingestors*, *AnalyticsComponents*, and *Sinks*. A *Component* exposes required and provided *Port* which works as the connection points. A *Port* contains a set of *Fields* which determines their data types for each component input and output. These data structures can be defined manually or inferred from the PMML data mining schema assigned to analytics components.

AnalyticsComponents are specialized in *Estimators* and *Transformers*, and they are the software component realizations of *PMMLModel* and *Transformations* respectively, and the PMML files define their behavior. An *Estimator* implements predictive algorithms supported by PMML such as bayesian networks, clustering, regression, decision trees, k-nearest, neural networks, vector machines, among others. While among the *Transformers* supported by PMML, we find normalization, discretization, value mapping, text indexing, aggregations, and built-in functions.

For more information about the PMML specification, please visit: http://dmg.org/pmml/v4-4-1/GeneralStructure.html.

The *Connectors* transfer data or control flow among components through input or output *Roles*. A set of connector types are defined based on the connector's classification proposed by Taylor et al. in [92]: *Procedure Call*, *Event*, *Stream*, *Adaptor*, *Distributor* and *Arbitrator*. A *Procedure Call* connector models the flow control and communication through invocations. Similarly, an *Event* connector affects the control flow and provides data transfer, but it is subject to the occurrence of events to notify all interested parts. A *Stream* connector is used to perform transfer of large amounts of data that is continuously generated. *Adaptors* enable interaction between components that have not designed to interoperate providing conversion features. *Distributor* connectors identify interaction paths and communication routing. An *Arbitrator* streamlines system operation and resolves conflicts thus offering intermediary services.

Component and *Connectors* define a set of additional properties required for implementation, such processing model (*procModel*: i.e., batch, micro-batch, stream), delivery guarantee (*delivery*: at-most-once, at-least-once, exactly-once, best-effort). Besides, a *Connector* specifies properties regarding their features taxonomy [92] such as synchronization type, notification, buffering, throughput, and protocol. *ArchDecisions*, from architectural drivers package, are mapped to components and connectors which are involved in such decision. This mapping allows ACCORDANT to trace the QS, architectural decision, and components/connectors to validate end-to-end design.

7.2.3 Deployment Viewpoint (DV)

The Deployment viewpoint integrates DevOps practices, including containerization, IaC, and serverless computing. The IT architect specifies how software artifacts (components and connectors) are deployed on a set of computation nodes in the ACCORDANT Task 3.2 using the DV. The DV metaclasses are inspired by container managers such Kubernetes[3]. The main metaclasses are detailed in Fig. 7.2.4.

The DV contains *Devices*, *Services*, *Deployments*, serverless environments (*ServerlessEnv*), and *Artifacts*. A *Device* is a worker machine (physical or virtual) on which the Pods are deployed. A *Pod* is a group of one or more execution environments (*ExecEnvironment*) which can share storage and network. An *ExecEnvironment* represents a container with a Docker image and specific resource requirements (CPU, memory). A *Deployment* specifies the desired state for a Pod's

[3]https://kubernetes.io

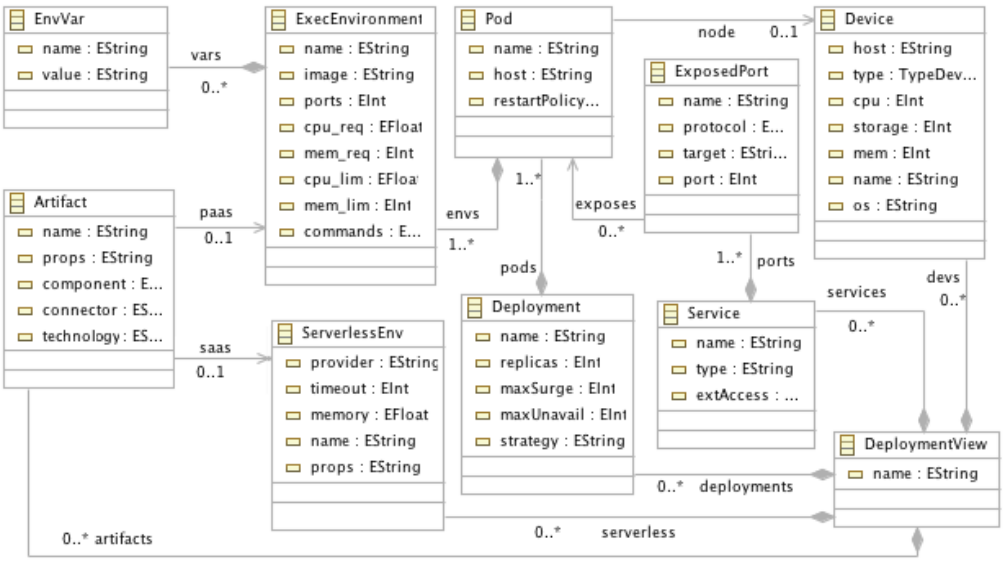

Figure 7.2.4: Excerpt of the Deployment viewpoint metamodel

group and its deployment strategy, including the number of replicas. *Services* and *ExposedPorts* define the policies, addresses, ports, and protocols by which to access Pods from outside the cluster network. A *ServerlessEnv* element describes a computing environment in which a cloud provider runs the server and dynamically manages the allocation of machine resources, as opposition to *ExecEnvironment* where physical resources have to be defined and managed.

DV metamodel comprises *Pod*, *ExposedPort*, and *Deployment* metaclasses to operationalize BDA applications in a specific infrastructure. It is noteworthy that a *FV* model can be deployed in different *DV* models either to use a different strategy or to test the fulfillment of predefined QS. An *Artifact* corresponds to an executable or deployable representation of functional elements (i.e. components and connectors from functional view) which can be deployed on either execution or serverless environments. An *Artifact* is associated to either *Component* or *Connector* from FV. To implement components and connectors, the IT architects defines a concrete technology via *Artifact*. This selection must match with available code generators templates. Architectural decisions (*ArchDecision*) can be assigned to *Deployments* and *Artifacts* to map explicitly deployment decision to achieve QS.

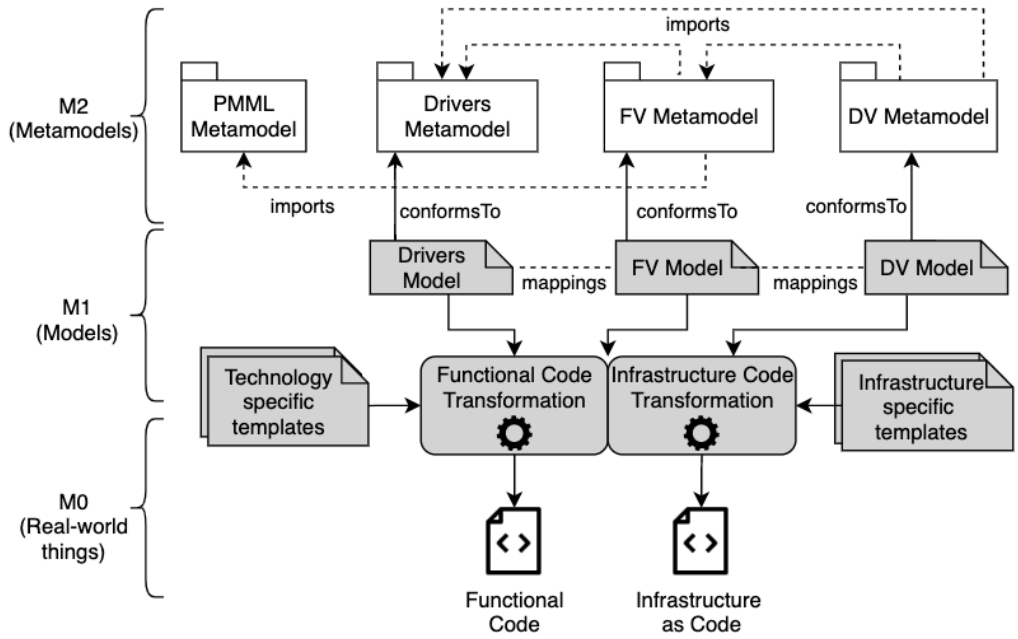

Figure 7.3.1: ACCORDANT's models transformations.

7.3 Code generation

Fig 7.3.1 details the transformation process from models and model-to-text templates. Once the IT architect specifies drivers, FV, and DV models, the ACCORDANT framework interweaves models using the inter-view mappings in activity 5 of the ACCORDANT Method. These mappings associate the PMML Model to FV elements (component and connectors), FV elements to deployment artifacts, and architectural decisions to FV and DV elements.

ACCORDANT performs code generation through model-to-text transformations contained in Xtend templates [4]. Code generation is twofold: functional and infrastructure (IaC) code; therefore, there are technology-specific templates for functional code (e.g., Apache Spark, Python, Java, etc.) and infrastructure-specific templates for IaC (i.e., Kubernetes).

7.3.1 Functional code generation

On the functional side, each component and connector is characterized by the features specified in the FV model (processing model, ML algorithm, delivery type, sync type) and the target technology selected in the DV. Functional code generators are implemented using Xtend templates for each concrete technology defined per software artifact, i.e., components and connec-

[4] https://www.eclipse.org/xtend

tors. These templates can be added to include new technologies. For instance, near real-time analytics requires stream or micro-batch processing offered by Apache Storm or Spark, respectively, and Event connectors such as Apache Kafka or RabbitMQ. Regarding the QS monitoring, code generators include specific instrumentation to log metrics at an application level. It allows us to collect specific QS from a high-level abstraction, saving the cost of adding code for logging metrics for each application and target technology.

A functional code transformation navigates the architecture drivers and FV models to generate code for each component and connector. A specific template is used according to the artifact technology assignment in the DV. Algorithm 1 describes how the functional code is generated for each component. This algorithm loops through each component, and depending on the metaclass, generates the code according to the technology template. When the component is an *AnalyticsComponent*, the PMML specification is used to load the file, build the model, and define input and output structure. In addition, the component's ports are also processed to generate the port field structure. Instrumentation code is generated to log QS metrics in the component whenever an architecture decision is associated.

Algorithm 2 describes the code generation for connectors regarding their metaclasses (*Event*, *ProcedureCall*). Each connector's role is also iterated to produce its code. In the case of connector of type *ProcedureCall*, a call invocation is generated in the client component that has the required port associated with that connector. Like components, if an architecture decision is associated with the connector, the instrumentation code is generated aligned to the QS.

A detailed example of an Xtend template that implements the functional code generation algorithm for Spark is illustrated in Listing 7.1. This template transforms each estimator component assigned to Apache Spark technology to a Spark driver and associates the PMML file to define its predictive behavior in lines 16-17. The component's data structure is inferred from the data dictionary specified in the PMML file in lines 22-24. This template also instruments the QS monitoring for each estimator component as detailed in lines 30 and 31 by calling the Prometheus client[5].

[5]https://prometheus.io

Algorithm 1: Functional code generation for components

```
input  : A functional model FV ≠ ∅
for comp ∈ FV.comps do
    template ← comp.artifact.technology
    if comp ∈ AnalyticsComponent then
        GeneratePMMLLoadingCode(comp.pmml, template)
        for port ∈ comp.ports do
            GeneratePortCode(comp.pmml.MiningSchema, template)
        end
    else if comp ∈ Ingestor then
        GenerateIngestionCode(comp, template)
        for port ∈ comp.ports do
            GeneratePortCode(port.fields, template)
        end
    else
        GenerateSinkExportCode(comp, template)
        for port ∈ comp.ports do
            GeneratePortCode(port.fields, template)
        end
    end
    if comp.decision ≠ ∅ then
        GenerateInstrumentationCode(comp.decision.QS)
end
```

Algorithm 2: Functional code generation for connectors

```
input  : A functional model FV ≠ ∅
for conn ∈ FV.conns do
    template ← conn.artifact.technology
    if conn ∈ Event then
        GenerateEventCode(conn)
        for role ∈ conn.roles do
            GenerateEventConnectionCode(role, template)
        end
    else if conn ∈ ProcedureCall then
        for role ∈ conn.roles do
            if role.port = REQUIRED then
                GenerateInvocationCallCode(role.port.comp)
            GenerateDeclarationCallCode(role.port.comp)
        end
    end
    if conn.decision ≠ ∅ then
        GenerateInstrumentationCode(conn.decision.QS)
end
```

7.3.2 Infrastructure as code generation

On the IaC side, the DV model is transformed into Kubernetes' configuration files (in YAML format) used to create and configure infrastructure over Kubernetes clusters. Kubernetes files contain *Nodes, Pods, Deployments,* and *Services* which are executed through the command *Kubectl*. Additional infrastructure templates could generate IaC for different deployments than Kubernetes, such as serverless or edge computing. Algorithms 3, 4, 5, and 6 describe the code generation for deployment, services, artifacts, and devices respectively.

Listing 7.1: Excerpt of the technology-specific transformation template.

```
1  def estimatorToJava(Estimator estimator) '''
2    /*
3     * generated by Accordant
4     */
5    package edu.uniandes.accordant.«estimator.funcView.name».estimator;
6    import org.apache.spark.SparkConf;
7    import org.jpmml.evaluator.Evaluator;
8    import io.prometheus.client.*;
9    ...
10   public class «formatJavaClassName(estimator.name)»Estimator {
11     public static void main(String[] args) throws Exception {
12       SparkConf conf = new SparkConf().setAppName("«estimator.name»Estimator")
13               .setMaster("local[*]");
14       JavaSparkContext sc = new JavaSparkContext(conf);
15       SparkSession sparkSession = new SparkSession(sc.sc());
16       InputStream pmmlFile = new URL("«estimator.pmml»").openStream();
17       EvaluatorBuilder evalBuilder = new LoadingModelEvaluatorBuilder().load(pmmlFile);
18       ...
19       Evaluator evaluator = evalBuilder.build();
20       TransformerBuilder builder = new TransformerBuilder(evaluator)
21             .withTargetCols().exploded(true);
22       List<StructField> fields = new ArrayList<StructField>();
23       «structField(estimator.pmml)»
24       StructType schema = DataTypes.createStructType(fields);
25       Transformer pmmlTransformer = builder.build();
26       Dataset<Row> resultDs = pmmlTransformer.transform(inputDs);
27       /*
28       * Metrics instrumentation
29       */
30       receivedBytes.observe(inputDs.size());
31       requestTimer.observeDuration();
32             ...
33       sparkSession.close();
34       sc.close();
35   '''
```

Algorithm 3: Infrastructure code generation for Devices

```
input  : A deployment model DV ≠ ∅
for dev ∈ DV.devs do
    GenerateDeviceCode(dev)
end
```

Algorithm 4: Infrastructure code generation for Deployments

```
input  : A deployment model DV ≠ ∅
for dep ∈ DV.deployments do
    GenerateDeploymentMetadata(dep)
    for pod ∈ dep.pods do
        GeneratePodCode(pod)
        for env ∈ pod.envs do
            GenerateEnvCode(env)
            for var ∈ env.vars do
                GenerateVarCode(var)
            end
        end
    end
end
```

Algorithm 5: Infrastructure code generation for Services

```
input  : A deployment model DV ≠ ∅
for serv ∈ DV.services do
    GenerateServiceMetadata(serv)
    for port ∈ serv.ports do
        GeneratePortCode(port)
    end
end
```

Algorithm 6: Infrastructure code generation for Artifacts

input : A deployment model $DV \neq \emptyset$

for $arti \in DV.artifacts$ **do**

  if $arti.component \neq \emptyset$ **then**

    $GenerateComponentGitClone(arti.component, arti.paas)$

  else if $arti.connector \neq \emptyset$ **then**

    $GenerateConnectorGitClone(arti.connector, arti.paas)$

  end

end

In Listing 7.2, an Xtend code fragment shows how each service in the DV is iterated over (line 4) and transformed into a Kubernetes service, specifying its name, type, and port attributes. For each service port in the DV model, a Kubernetes port, protocol, and target are generated (lines 15 to 18).

Listing 7.2: Excerpt of the infrastructure-specific transformation template.

```
1  def serviceToKubeService(DeploymentView deployView)
2          '''
3          # generated by Accordant
4          «FOR serv : deployView.services»
5          apiVersion: v1
6          kind: Service
7          metadata:
8            name: «format(serv.name)»
9          spec:
10           type: «serv.type»
11           selector:
12              component: «format(serv.name)»
13         «IF serv.ports!==null»
14                   ports:
15                 «FOR port: serv.ports»  - name: «port.name»
16                     protocol: «port.protocol»
17                     port: «port.port»
18                     targetPort: «port.target»
19                 «ENDFOR»
20         «ENDIF»
21         ---
22
23         «ENDFOR»
24
25         '''
```

7.4 Summary

This chapter presents the ACCORDANT MF and its main components. We described the main packages and detailed the metamodels which formalize BDA applications in drivers, FV, and DV. We also detailed the code generation process, which produces both functional and infrastructure code from ACCORDANT models. This DSM approach allows us to design an integrated BDA architecture driven by the QS and generate functional and infrastructure code from model definitions. The code generation enables ACCORDANT to instrument monitoring to validate components' performance against the initial QS. In the following chapter, we will address the model validation against architectural constraints.

8

Architectural validation in big data analytics applications

This chapter[1] presents a framework based on the ACCORDANT MF and an ontology to specify constraints and validate them over BDA applications. This chapter covers Activity *4- Validation* in the ACCORDANT Method. Our validator addresses quality of service, infrastructure, and technology mismatches expressed as inter and intra-view constraints using the Object Constraint Language (OCL) and a semantic query language. This chapter aims at developing Research Objective *RO7*.

8.1 Introduction

Garlan et al. in [37] address the architectural mismatch problem by classifying architectural assumptions in categories and component interaction facets which can lead to mismatches: assumptions about the application domain, components-connectors interaction, and infrastruc-

[1]Portions of this chapter have been submitted to: Castellanos, B. Perez, C., Varela, C. A., Correal, D. (2021) "Architectural Mismatch in Big Data Analytics Applications: A Domain-Specific Model Approach". In Journal of Systems and Software.

ture. Besides, Garlan et al. propose some techniques to prevent, detect, and repair mismatches, such as architectural specialization, standardization, tool-assisted detection, and software process guidance.

In Chapter 4, the review of the related work found that previous studies fall short of validating or verifying BDA architecture and applications. Previous works have proposed formal ADLs to specify software architectures and to detect structural, behavioral [12, 75], and quality of service [105] violations. The survey presented in [103] found that the lack of specific approaches and tools are the significant challenges for identifying and recording architectural assumptions. Previous ADLs are too general or fall short of detecting and preventing architectural mismatch from different viewpoints regarding the BDA domain. For instance, they do not consider the data processing model (streaming, batch), delivery semantics (at most once, at least once, exactly once), technology, and deployment mismatches. Besides, current ADLs on BDA are too generic to support tool-assisted detection of architectural mismatch.

This chapter presents an architectural mismatch framework among architectural views to specify and detect mismatches on BDA architectures by formalizing them as domain-specific constraints. This formalization makes explicit the architecture assumptions and allows us to validate them automatically. This validation corresponds to Activity *4- Validation* in the ACCORDANT Method. We address functional, quality, infrastructure, and technology mismatches by specifying inter-view and intra-view constraints using OCL [93] on the ACCORDANT metamodels. Once inter-view and intra-view constraints are specified, they can be evaluated automatically over architectural models conform to ACCORDANT metamodel. We extend the ACCORDANT metamodel with well-formed OCL constraints making architectural assumptions explicit to support automated mismatch checking. In summary, the contributions of this Chapter are:

- A metamodel to describe BDA architectural drivers, design, and constraints.
- An ontology to specify BDA technologies and validate their consistency.
- A set of architectural mismatch definitions expressed as inter-view and intra-view constraints.
- A set of constraint specifications expressed in OCL and a semantic query language.

The rest of this chapter is organized as follows. Section 8.2 reviews the work related with architecture validation. Section 8.3 presents our approach based on ACCORDANT. Section 8.4 summarizes this chapter.

Table 8.2.1: Related Work

Work	Approach	Domain	Spec Level	Validations				
				Reliability	QS	Deployment	Domain Features	Technology Compliance
π-AAL [67]	FOPL	General	S,B	✓				
π-ADL with QoS [105]	FOPL	General	S,B	✓	✓			
Bernardo et al. [12]	PA	General	S,B	✓				
IVML [34]	FM	BDA	S,B	✓			✓	
Zosas et al. [106]	FM	BDA	S				✓	
D-VerT [64]	LTL	BDA	S,B		✓			
BIGOWL [7]	OWL	BDA	S				✓	✓
BiDArch [90]	OWL	BDA	S				✓	✓

8.2 Related Work

In Chapter 4, we reviewed the related work about BDA development and deployment, and we found little research about architecture validation. Therefore, in this section, we deepen our review focused on architecture validation studies that can support the detection of architectural mismatch in BDA. Table 8.2.1 outlines the reviewed works along with their main characteristics and supported validations. The review include domain general ADLs [67] [105], [12] and BDA-specific frameworks [34], [106], [7], [90], [64]. This comparison highlights how our approach focuses its contribution by covering functional reliability (comprising consistency, correctness, and reachability), quality of service (QS), deployment, domain features, and technology compliance validations.

Previous works have proposed formal ADLs to specify software architectures to check structural and behavioral reliability (consistency, correctness, and reachability) and detect QoS violations. Mateescu and Oquendo introduce πAAL in [67], an architecture analysis language based on π-calculus for specifying dynamic and mobile architectures. πAAL provides a framework for defining software architectures' properties: structural (e.g., cardinality and interconnection topology) or behavioral (e.g., safety, liveness, and fairness). Architectural properties are specified in terms of logical formulas checked against the architecture description. Bernardo et al. in [12] detect three causes of mismatches in software systems: incompatibility between two components with several interactions and the lack of interoperability among a set of components forming a cyclic topology. Their proposal checks architectural compatibility to detect deadlocks. Zhou and Li in [105] extend πADL with Quality of Service (QoS) constructs to fa-

cilitate the modeling of QoS-aware applications. These QoS constructs allow checking and detecting QoS violations at the architectural design stage. Although these proposals address architectural mismatch detection focused on structural, behavioral, and quality of service facets, they are too general. They fall short of detecting architectural mismatches in different viewpoints in the BDA domain. Specifically, they do not consider data processing models (streaming, batch), delivery semantics (at most once, at least once, exactly once), technology, and deployment mismatch.

Within the BDA domain, the architectural mismatch has been discussed in [34, 64]. Marconi et al. [64] present an approach for non-functional analysis through D-VerT. D-VerT performs an architectural assessment based on a translation of Storm topologies into a Linear Temporal Logic (LTL) metric to check components that cannot process their workload on time. Eichelberger et al. [34] deal with the variability in the topology of BDA applications' product lines using Integrated Variability Modeling Language (IVML). A BDA application is modeled using IVML, specifying the pipeline structure, analytics algorithm family, data sources, and infrastructure. Besides, an IVML model includes topological constraints to ensure complex properties evaluated using first-order logic. Those proposals are focused on BDA applications, but they only verify time-related or structural properties, and they do not consider processing semantics, technology, and infrastructure mismatches. Zozas et al. [106] proposes a feature model to design BDA applications, with features ranging from infrastructure (network, storage, and data sources) to services (collect, store, processing, and user interfaces). Their feature model includes the variability and constraint relationships, and UML models derive from specific product configurations.

Some works have tackled BDA from an ontological perspective by defining attributes, relationships, and individuals in this domain. Barba-González et al. propose BIGOWL [7], an abstract top-level ontology, to support knowledge management in BDA applications. BIGOWL includes concepts like components (collection, processing, and sink), algorithms, and data orchestrated within workflows. BIGOWL recommends BDA components and implementations based on initial configurations, and then the workflow is designed and refined using queries interactively. Similarly, BiDArch [90] proposes a BDA ontology to automatically establish a feasible architecture from the problem characteristics and technology tools properties. This feasibility is determined using a semantic catalog with the Hadoop ecosystem tools and a reference model to recover valid architectures for the characteristics of a defined BDA problem.

8.3 Detecting architectural mismatch using ACCORDANT

The ACCORDANT framework includes domain-specific constraints to make assumptions explicit and detect architectural mismatches. Our approach addresses three of the methods presented by Garlan et al. in [37] to detect architectural mismatches: i) Work in an architecturally specialized domain, in our case, applications scoped in the BDA domain. ii) Standardize documentation to make assumptions and constraints explicit by using a domain-specific ADL. iii) Offer process guidance for building early prototypes to expose mismatches by providing tool-supported validation and code generation to improve the software development process.

We have presented architecture models validation in Activity 4 of the ACCORDANT method in Fig. 5.2.1. In this activity, ACCORDANT FV and DV models are validated against the OCL constraints defined within each view or among views. Fig. 8.3.1 details the interaction of ACCORDANT metamodel packages and the OCL validations defined within each view (termed as *intra-constraints*) and between different views (*inter-constraints*). This domain-specific modeling and OCL approach aims at making architectural assumptions explicit and formal to validate the consistency of three facet interactions proposed in [37]:

- *Component-connector interaction*: FV models and *intra-FV* constraints of component and connectors within a BDA pipeline.
- *Infrastructure*: DV models and *intra-DV* constraints of infrastructure and deployment artifacts.
- *Application domain*: Inter-model mappings and inter-view constraints to specify and restrict interactions among different views, i.e., architectural drivers, FV, DV, and technology.

In addition to the ACCORDANT metamodel, we extend a BDA ontology to define classes, individuals, and reasoning on them semantically. Specifically, we extend a previously presented BDA ontology BIGOWL [7], by adding connectors and technologies to check machine learning algorithms support and technology compatibilities.

8.3.1 The ACCORDANT ontology: extending BIGOWL

Fig. 8.3.2 offers an overview of ACC-OWL, highlighting the main classes and its hierarchy developed in this work within the red dotted frame. The classes outside the red dotted frame were previously defined in BIGOWL, and they have relationships with ACC-OWL elements. The main

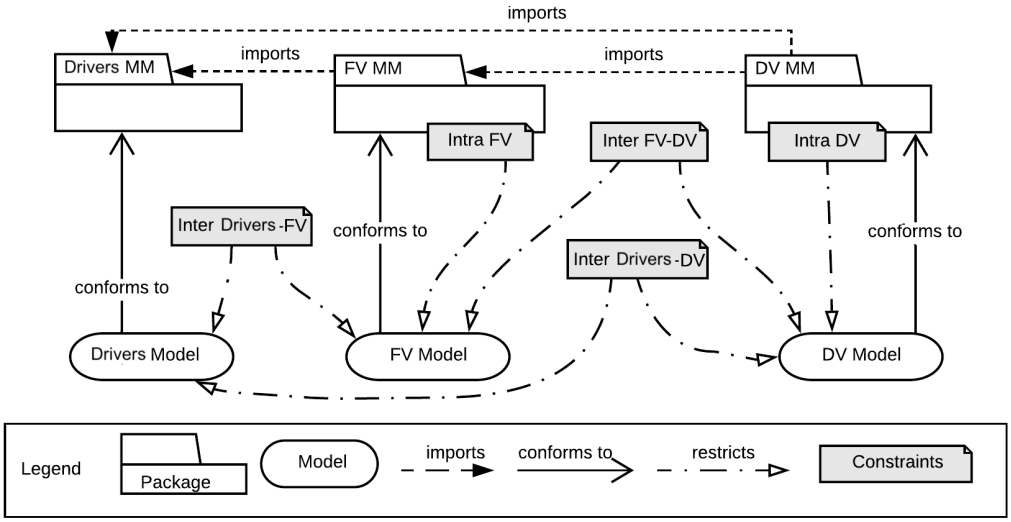

Figure 8.3.1: Architectural mismatch validation using OCL constraints

ACC-OWL extensions in this paper address the architectural point of view of BDA pipelines, and they are detailed in Table 8.3.1. Our extension sums up 17 classes, 6 object properties, 33 individuals, and 3 axioms.

The *Connector* ontology class is analogous to the Connector metaclass in the ACCORDANT FV. *Technology* catalogs the available tools that can be used to implement BDA components or connectors, and we have added individuals which cover open source tools or libraries such as Apache Flink, Flume, Hadoop, Hive, Kafka, Samza, Spark, Sqoop, Storm, RabbitMQ, Scikit-learn, and specific connectors like Hive Streaming, MongoSpark. A *TechFeature* specifies a characteristic that is offered or supported to implement components or connectors in a specific BDA technology. In our case, we include two features: processing model (batch, micro-batch, stream) and delivery semantics (at least once, at most once, exactly once, best effort). The *License* class defines the use permissions/restrictions for a specific technology. This specification enables us reasoning about when architectural constraints include aspects like proprietary or non-proprietary licenses. In addition, we have included some object properties such *techCompatibleWith* to state which technology is directly compatible with other technology without additional coding, and *supportAlg* to define that a specific technology supports a built-in ML algorithm.

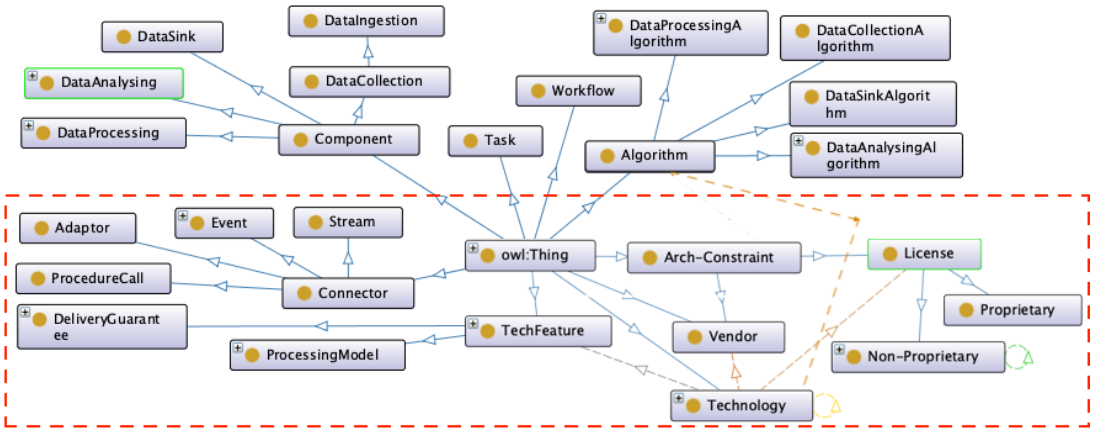

Figure 8.3.2: Accordant ontology extension

Table 8.3.1: Extract of ACC-OWL extensions

Name	Type	Description
Connector	Class	Communicate two components
Adaptor	Class	Subclass of Connector which corresponds to Adaptor metaclass
Event	Class	Subclass of Connector which corresponds to Event metaclass
Stream	Class	Subclass of Connector which corresponds to Stream metaclass
ProcedureCall	Class	Subclass of Connector which corresponds to ProcedureCall metaclass
TechFeature	Class	Specifies a technology feature (e.g., delivery guarantee, processing model)
Arch-Constraint	Class	Architecture constraint
License	Class	Technology license type (e.g., Propietary, Copyleft, LGPL, etc) subclass of Arch-Constraint
Vendor	Class	Technology Vendor subclass of Arch-Constraint
techCompatibleWith	Object Property	A technology is compatible with each other without adaptors: *Technology* $\Longrightarrow$ *Technology*
hasTechFeature	Object Property	A technology has a feature: *Technology* $\Longrightarrow$ *TechFeature*
supportAlg	Object Property	A technology supports an analytics algorithm: *Technology* $\Longrightarrow$ *Algorithm*

Table 8.3.2: Intra-view constraints. (*) Constraints illustrated in detail

Code	Description	Property	View	Impl
F1	All BDA pipelines must have some ingestor	Completeness	FV	OCL
F2	All BDA pipelines must have some sink	Completeness	FV	OCL
F3	All ingestor components must have some provided port and no required port	Completeness	FV	OCL
F4	All sink components must have some required port and no provided port	Completeness	FV	OCL
F5	All analytics components must have provided and required ports	Completeness	FV	OCL
F6	All connectors must have some input and output roles	Completeness	FV	OCL
F7	All components must be connected to connectors	Completeness	FV	OCL
F8	All connectors must be connected to components	Completeness	FV	OCL
F9	None of the components can have self-connections	Consistency	FV	OCL
F10	None of the components can be in a cyclic connection	Consistency	FV	OCL
F11	All components' processing models must match	Consistency	FV	OCL
F12*	All components and connectors delivery guarantee must match	Consistency	FV	OCL
F13	All input roles must be connected to provided port, and output roles to required port	Consistency	FV	OCL
F14*	All connected ports must be compatible	Consistency	FV	OCL
D1	All technology dependencies must match with deployment artifacts	Consistency	DV	SPARQL
D2	All artifacts must be deployed	Completeness	DV	OCL
D3*	All device resources must support the pods' requirements	Consistency	DV	OCL

8.3.2 Intra-view constraints

This section details how the architectural mismatch detection is implemented within the functional (FV) and deployment views (DV). These intra-view constraints are specified as invariants in OCL or SPARQL queries over ACC-OWL declared among concepts within the same view. Table 8.3.2 details the intra-view constraints for FV and DV defined in this research. This constraints list is not intended to be comprehensive but representative of the BDA domain. In this table, each constraint is identified by a *Code*, *Description*, the *Property* to be validated, *View*, and its implementation language (*Impl*). The star symbol (*), next to the code, identifies the constraints that will be illustrated in detail in this document. We choose these constraints to be

explained and detailed because the mismatches covered by them were actually experienced during the design and implementation of the case studies that will be presented in Section 11.4. All constraint implementations can be found at the ACCORDANT project repository[2].

Listing 8.1 details the definition of intra-constraints *F12*, *F14*, and *D3* as OCL invariants (*inv*). Since OCL expressions can be equivalent to first order logic (FOL), we describe the corresponding representations in first order logic for better readability and understanding. Constraint *F12*, lines 1-3, states that all connected components' delivery guarantee (exactly once, at most once, and at least once) match. In FOL, *F12* is defined as: $\forall c1, c2 \in Components \wedge Connected(c1, c2) \wedge c1 <> c2 \implies delivery_{c1} = delivery_{c2}$. Invariant *F14* checks that all connected ports' types must be compatible, so lines 5-7 specify: $\forall p0, p1 \in Ports \wedge Connected(p0, p1) \wedge p0 <> p1 \implies fieldsSetMatch\ (p0, p1)$. Predicate *fieldsSetMatch*, lines 9-12, compares the ordered sequence of fields' data types for each port. Finally, constraint *D3* holds that the sum of all device resources (memory and CPU) must be less than or equal to the device's available resources. Lines 14-16 states: $\forall d \in Devices \wedge p \in d.pods \wedge (\sum_{n=1}^{d.pods} cpu_n) \leq cpu_d \wedge (\sum_{n=1}^{d.pods} memory_n) \leq memory_d$. Both predicates $\mathrm{getTotalReqCPU}$ and $\mathrm{getTotalReqMemory}$ (lines 18-22) take into account the product of resource required values and the number of pod replicas.

Listing 8.1: Extract of intra-view constraints in OCL

```
1  context Connector inv F12:
2  self.roles.port.comp->forAll(c1, c2 | c1 <> c2
3  implies c1.delivery = c2.delivery)
4
5  context Port inv F14:
6  self.connectedPorts()->forAll(p1 | self <> p1
7  implies p1.fieldsSetMatch(self))
8
9  def: fieldsSetMatch(p1 : Port):
10 Boolean = self.fields->sortedBy(order)->
11 collect(f : Field | f.dtype) = p1.fields->
12 sortedBy(order)->collect(f1 : Field     | f1.dtype)
13
14 context Device inv D3:
15 self.pods.getTotalReqCPU()->sum()<= 95% and
16 self.pods.getTotalReqMemory()->sum() <= self.mem
17
18 context Pod def: getTotalReqCPU(): Real=
19 self.envs.cpu_req->sum()*self.deplOwner.replicas
20
21 context Pod def:getTotalReqMemory(): Integer =
22 self.envs.mem_req->sum()*self.deplOwner.replicas
```

[2]https://github.com/kmilo-castellanos/accordant

8.3.3 Inter-view constraints

This section details the inter-view constraints, which cross multiple views, i.e., architectural drivers, functional (FV), and deployment view (DV). These constraints validate consistency at different levels of abstraction by comparing architectural drivers with component-connector, deployment, and technology models. Table 8.3.3 details inter-view constraints specifying involved views and their implementation. Like the intra-view constraints, this list is not intended to be comprehensive but representative of the BDA domain.

Listing 8.2 details the definition of inter-constraints *IF1* and *ID5* as OCL invariants, and Listing 8.3 explains the definition of inter-constraint *FD3* and *FD5* as SPARQL queries. Constraint *IF1* in Listing 8.2 (lines 1-4) states that all components, defined in the FV, must match with processing models constraints specified as architectural drivers, hence aligning architectural design and requirements. In FOL, *IF1* is defined as: $\forall comp \wedge comp \in Components \wedge const \in Constraints \wedge isProcModel(const) \implies procModel_{comp} \in Constraints$

Listing 8.2: Extract of inter-view constraints in OCL

```
1  context FunctionalView inv IF1:
2  self.project.constraints->select(type=PROC_MODEL)
3  ->asSet() -> includesAll(self.comps->
4  collect(procModel)->asSet())
5
6  context DeploymentView inv ID5:
7  self.artifacts -> forAll(a1,a2:Artifact| a1<>a2
8  and a1.comp.provides(a2.comp) implies a1.supportsQS(a2))
9
10 context Artifact def: supportsQS(a2:Artifact):
11         Boolean = comp.decision.aqs.qs.measure = a2.comp.decision.
12         aqs.qs.measure
13         implies comp.decision.aqs.qs.minValue * paasEnv.podOwner.
14         deplOwner.replicas * comp.decision.aqs.qs.measure.order
15         >= a2.comp.decision.aqs.qs.minValue * a2.paasEnv.podOwner.
16         deplOwner.replicas * a2.comp.decision.aqs.qs.measure.order
```

Constraint *ID5*, lines 6-8, specifies that all provided component's quality scenarios response measure must support (be greater than or equal to) required component's quality scenario. This constraint includes the number of replicas of the deployed artifact, in DV, associated to the component in FV. *ID5* is defined as:

$\forall a1, a2 \in Artifacts \wedge a1 <> a2 \wedge Provides(component_{a1}, component_{a2}) \implies SupportsQS(a1, a2)$.

The predicate *Provides* checks if $component_{a1}$ has a provided port connected to $component_{a2}$. Predicate *SupportsQS* (lines 10-16) checks if QS of two artifacts have the same measure (e.g.,

Table 8.3.3: Inter-view constraints. (*) Constraints illustrated in detail

Code	Constraint	Property	Views	Impl
IF1*	All components must match with processing models constraints	Consistency	Drivers-FV	OCL
IF2	All significant architectural decisions must be reflected in components-connectors	Completeness	Drivers-FV	OCL
IF3	All components must match with delivery guarantee constraints	Consistency	Drivers-FV	OCL
ID1	All selected technologies must match with license constraints	Consistency	Drivers-DV	SPARQL
ID2	All selected technologies must match with model processing constraints	Consistency	Drivers-DV	SPARQL
ID3	All selected technologies must match with delivery guarantee constraints	Consistency	Drivers-DV	SPARQL
ID4	All deployments must match with cloud models constraints	Consistency	Drivers-DV	OCL
ID5*	All provided QS must match with required QS between connected components	Consistency	Drivers-DV	OCL
ID6	All significant architectural decisions must be reflected in deployment artifacts	Completeness	Drivers-DV	OCL
ID7	All selected technologies must match with vendor constraints	Consistency	Drivers-DV	SPARQL
ID8	All metrics of deployed artifacts must match with quality scenarios measures	Correctness	Drivers-DV	Ad Hoc
FD1	All technologies features must match with components and connectors properties	Consistency	FV-DV	SPARQL
FD2	All components and connectors must be mapped to deployment artifacts	Completeness	FV-DV	OCL
FD3*	All connected component and connectors technologies must be compatible	Consistency	FV-DV	SPARQL
FD4	All component-connector properties must be satisfiable by available technologies	Consistency	FV-DV	SPARQL
FD5*	All Estimator algorithms must be supported by available technologies	Consistency	FV-DV	SPARQL

throughput, availability, etc.) and implies that the first artifact's (*a1*) QS response measure must be greater than or equal to the second artifact's QS response measure, taking into account the artifact's numbers of replicas. The property *order* of *measure:QAMetric* specifies if the measure value is comparable in ascending (1) or in descending order (-1).

Listing 8.3: Examples of inter-view constraints in SPARQL

```
1  PREFIX bigowl: <http://www.khaos.uma.es/perception/bigowl#>
2  PREFIX accowl: <http://www.uniandes.edu.co/accordant/technology#>...
3  #FD3
4  SELECT DISTINCT ?tech1 ?tech2 WHERE {
5    ?tech1 rdf:type accowl:Technology
6  . ?tech2 rdf:type accowl:Technology
7  . ?tech1 accowl:techCompatibleWith ?tech2
8  . FILTER ( ?tech1=?param1 && ?tech2=?param2 ) }
9  #FD5
10 SELECT DISTINCT ?tech ?model WHERE {
11   ?tech rdf:type accowl:Technology
12 . ?model rdf:type bigowl:Algorithm
13 . ?tech accowl:supportAlg ?model
14 . FILTER ( ?tech=?param1 && ?model=?param2 ) }
```

Technology compatibility constraints are defined using SPARQL queries over the ACCOWL ontology, such as *FD3* and *FD5* detailed in Listing 8.3. *FD3* states that all connected components and connector technologies must be compatible. *FD3* is implemented as a SPARQL query, lines 4-8. SPARQL query *FD3* states that given two technologies (*tech1* and *tech2*) which corresponds to the parameters *param1* and *param2* (technology names), both technologies must be related through the object property *accowl:techCompatibleWith*. These query's parameters are the artifact's technologies in the DV (*Artifact::technology*), which are connected in the component-connector model in FV. *FD5* validates that all estimator's machine learning algorithms must be supported by their assigned technology, lines 10-14.

8.4 Summary

We have presented a framework to specify, detect, and prevent architectural mismatches on BDA architectures. We have presented a set of inter- and intra-view constraints, an ACCORDANT metamodel extension, a BIGOWL ontology extension, and OCL specifications to implement these constraints.

The intra-view constraints allow us to verify consistency, completeness, and correctness among components within the same view, while inter-view constraints apply among views. These constraints can check violations at functional, deployment, infrastructure, and technology

levels in the early development stages. Regarding the inter and intra-view constraints, related work can be classified to determine their coverage regarding architectural views and validated properties. Our thesis has presented a broader set of consistency and completeness validations than previous works by including structural and QS, deployment, infrastructure, and technology facets.

9

The ACCORDANT Method

In previous chapters, we have detailed the ACCORDANT MF and Validator. In this chapter[1], we revisit the ACCORDANT Method presented in Section 5.2 detailing each activity using the running example introduced in Section 1.3. The ACCORDANT Method addresses specifically the research objective *RO4*.

The ACCORDANT method is cyclic, iterative, and composed of seven activities across four software development life-cycle phases as depicted in Fig. 9.0.1: Requirements elicitation, development, deployment, and operation. We align the ACCORDANT method to two other well-known methods to design and evaluate software architecture. Our purpose is to design and evaluate BDA architectures by quality scenarios (QS) in an integrated way using ACCOR-

[1]Portions of this chapter previously appeared in: Castellanos, C., Varela, C. A., Correal, D. (2021). "ACCORDANT: A Domain Specific-Model and DevOps Approach for Big Data Analytics Architectures". Journal of Systems and Software, 172, 110869. C. Castellanos, B. Perez, D. Correal and C. Varela, "A Model-Driven Architectural Design Method for Big Data Analytics Applications". In 2020 IEEE International Conference on Software Architecture Companion (ICSA-C), Salvador, Brazil, 2020 pp. 89-94. Castellanos, C., Varela, C. A., Correal, D. (2019, September). "Measuring performance quality scenarios in big data analytics applications: a DevOps and domain-specific model approach". In Proceedings of the 13th European Conference on Software Architecture-Volume 2 (pp. 165-172). Castellanos C., Correal D., Rodriguez JD. (2018) "Executing Architectural Models for Big Data Analytics". In: Cuesta C., Garlan D., Pérez J. (eds) Software Architecture. ECSA 2018. Lecture Notes in Computer Science, vol 11048. Springer, Cham.

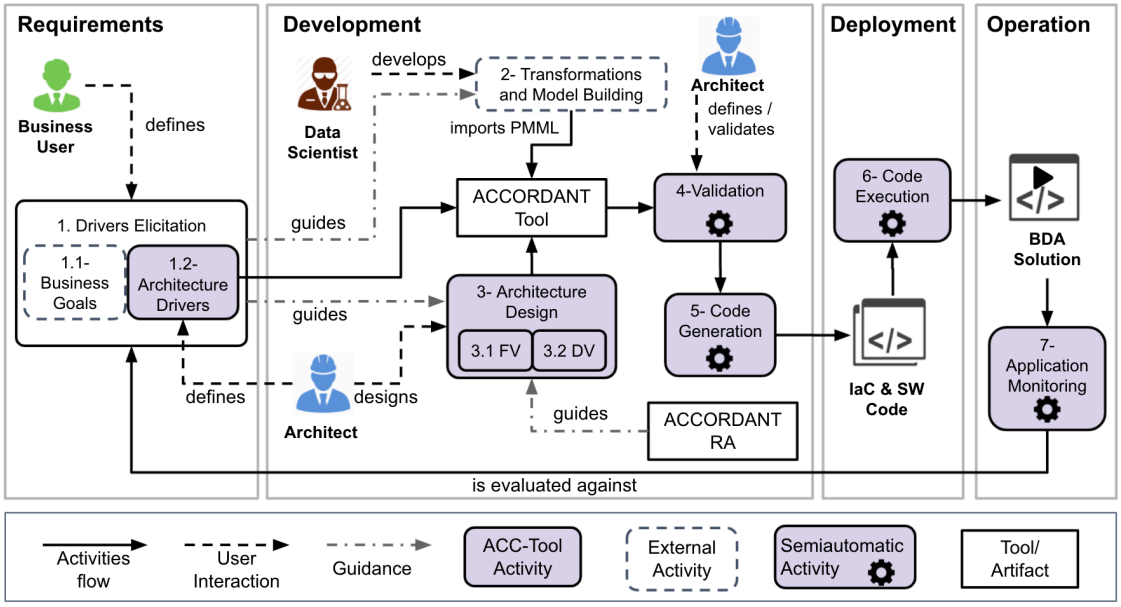

Figure 9.0.1: The ACCORDANT Method

DANT and following well-established methods. Thus, the deployment of the application designed in ACCORDANT can also be evaluated based on the design drivers. The activities in the ACCORDANT Method can be composed of tasks. The activities and tasks framed in solid lines are performed using the ACCORDANT MF directly, while activities and tasks made with external tools are framed in dotted lines. The activities marked with a gear icon are executed semi-automatically by ACCORDANT (see activities 4, 5, 6, and 7).

9.1 Activity 1 - Drivers elicitation

This activity comprises two tasks: the definition of business goals and architecture drivers. These drivers will guide the following activities.

9.1.1 Task 1.1 - Define business goals

Business users define the purpose of the BDA application in terms of business objectives. The business goals lead the analytics tasks, applications design, and the expected outcomes of the overall solution. For instance, a business goal in a telecom company could be:

Business goal 1: *Increase the customer retention indicators by identifying customers at high risk of churn and implementing actions to retain them.*

9.1.2 Task 1.2 - Define architecture drivers

The business user and IT architect define QS, and the architect can add some constraints that the architecture must fulfill. QS describes the BDA application quality goals in terms of quality attributes measured in quantifiable metrics. Continuing with our running example, to integrate real-time indicators with the new churn estimation service and implement customer retention actions quickly, we define the following QS:

QS 1: *The average response time for a request to the churn estimator system must be lower than 2 seconds under normal system operation.*

In addition, the architect can add constraints to validate the general system's requirements aligned to the IT policies. For instance, in the churn estimation project, the IT architect could establish:

Constraint 1: *All application components must be implemented using open source technologies to avoid licensing costs.*

Constraint 2: *This application shall use a streaming or micro-batch processing model since the batch processing model is not adequate regarding the QS 1.*

The IT architect specifies both QS and architectural drivers within a project called *Churn-Project* as a model conforms to the corresponding metamodel. Listing 9.1 provides the formal definition of the architectural drivers using the DSL of the ACCORDANT MF presented in Chapter 7. From lines 3 to 9, the QS1 specifies the quality attribute (QA), stimulus, environment, response, and the response measure. The response measure is defined in line 8 through a metric associated with the QA (latency), a range (0.0-2.0), and metric unit (seconds) to enable ACCORDANT to monitor this metric. Reviewing the ACCRA's tactics catalog, the architect decides to use the increased available resources and parallel processing tactics to improve the performance. Hence, the architecture driver specification also includes the QS analysis (lines 12-20) and decision (*D1*) of using those tactics to achieve the expected QS metric in lines 15-19. Finally, general constraints Constraint1 and Constraint2 that apply to the whole solution are stated in lines 23 and 24. These constraints will be later validated in Activity 4.

Listing 9.1: Architecture drivers specified using ACCORDANT MF

```
1  Project ChurnProject{
2    QScenarios {
3      QS QS1 {
```

```
4          QA:PERFORMANCE
5          stimulus: "An incoming customer churn estimation request"
6          environment: "Normal mode"
7          response: "The system estimates the customer churn likelihood"
8          measure: LATENCY between 0.0 and 2.0 in SECONDS
9        }
10     }
11     analyzedQScenarios{
12       AQS AQS1 of QS1 {
13         reasoning :"QS limits to 2 sec the latency to provide real time estimation"
14         ArchDecisions{
15           Decision D1{
16             isSensitivityPoint
17             rationale: "To introduce concurrent/parallel processing ..."
18             code: "D1"
19             tactics (IncreaseAvailableResources, ParallelProcessing)
20           }
21     ...
22     constraints{
23       Constraint {type: PROC_MODEL = "MICROBATCH"},
24       Constraint {type: LICENSE = "OPENSOURCE"}
25     }
26   }
```

9.2 Activity 2 - Data transformations and analytics model building

Data scientists select the algorithms, develop the data transformations, build and evaluates the analytics models guided by the business goals. In this case, given the business goal 1, the analytics task corresponds to a binary classification problem to predict a churn score (1, 0). So, data scientists explore, clean, and integrate customer data to train a model for binary classification, for instance, using a decision tree algorithm. This activity is made using external tools, the tools that are frequently used in the data lab environment, such as we identified in the BDA practice (see Chapter 3). Our approach in this activity is allowing data scientists to use the tools and frameworks that are natural for them without the concerns related to deploying the application aligned to QS in different technologies.

Compared to current approaches, the data scientist produces technology neutral artifacts (PMML specifications) to be integrated into the BDA application instead of generating source code as the output of the analytic process. Our approach allows data scientists to use their tools and decouple the analytics development outputs (i.e., predictive models) from the software development process. As mentioned in Chapter 2, PMML is an open standard that enables transformations and analytics models to be interchanged using technology-neutral formats such as XML. There is a wide range of tools to develop analytics models and transformations and ex-

port them to PMML[2].

As an example, Listing 9.2 shows an extract of the Python code to train a decision tree model to predict customers' churn probability using the Scikit-learn library. Lines 1 to 3 import the required libraries. Lines 5 and 6 define the data columns and read historical customer data to train the model. Lines 6 and 7 assign the dataset, independent variables to x and the dependent or target variable (Churn) to y. Lines 9 and 10 instantiate a decision tree classifier and fit the model to the dataset. Line 12 and 13 apply cross-validation and calculate the model scores mean to evaluate the model accuracy. Once the data scientist iterates the model to achieve the expected accuracy, the resulting model is exported to a PMML file in line 15.

Listing 9.2: Extract of Python code to train a decision tree model using Scikit Learn

```
1  from sklearn import tree
2  from sklearn2pmml import sklearn2pmml
3  import pandas as pd
4  ...
5  features = ['VMailMessage',..., 'Churn']
6  data = pd.read_csv('customersData.csv', names=features)
7  x = data[features[:-1]]
8  y = data['Churn']
9  clf = tree.DecisionTreeClassifier()
10 clf = clf.fit(x, y)
11 ...
12 scores = cross_val_score(clf, x, y, cv=5)
13 print(scores.mean(), scores.std())
14 ...
15 sklearn2pmml(estimator=clf, mapper=d_mapper, pmml='ChurnModel.pmml')
```

As a result, the PMML file contains the trained model in a technology-neutral format which can be implemented in multiple target technologies. Listing 9.3 details and extract of the *ChurnModel.pmml* file exported from the Python code. The Data Dictionary section (lines 2 to 11) specify each data field that comprises the dataset input, specifying data types, operational types, intervals, value maps, among others. Lines 12 to 27 define the tree model, including the mining schema and the nodes hierarchy, which accounts for the distribution for each tree node.

Listing 9.3: Extract of the ChurnModel.pmml file

```
1  <PMML version="4.1" xmlns="http://www.dmg.org/PMML-4_1" ... >
2    <DataDictionary numberOfFields="21">
3      <DataField name="VMailMessage" optype="continuous" dataType="integer">
4        <Interval closure="closedClosed" leftMargin="0.0" rightMargin="51.0"/>
5      </DataField>
6      ...
7      <DataField name="Churn" optype="categorical" dataType="string">
```

[2]dmg.org/pmml/products.html

```
8         <Value value="0"/>
9         <Value value="1"/>
10      </DataField>
11    </DataDictionary>
12    <TreeModel modelName="DecisionTree" functionName="classification" ...>
13      <MiningSchema>
14          <MiningField name="VMailMessage"/>
15          ...
16          <MiningField name="Churn" usageType="target"/>
17      </MiningSchema>
18      <Node id="0" score="0" recordCount="2666.0">
19          <True/>
20          <ScoreDistribution value="0" recordCount="2280.0"/>
21          <ScoreDistribution value="1" recordCount="386.0"/>
22              <Node id="2" score="0" recordCount="2502.0">
23              <SimplePredicate field="DayCharge" operator="lessOrEqual" value="44.96"/>
24              <ScoreDistribution value="0" recordCount="2216.0"/>
25              <ScoreDistribution value="1" recordCount="286.0"/>
26              ...
27      </TreeModel>
28  </PMML>
```

9.3 Activity 3 - Software architecture design

In this activity, the IT architect analyzes and designs the BDA architecture using the ACCORDANT MF and guided by business goals and the ACCORDANT reference architecture. Patterns and tactics catalog defined in the reference architecture provides generic architectural mechanisms to be instantiated on concrete architectures. The architect also formally integrates the architecture drivers defined in the *Task 1.2* by associating architectural decisions to architecture elements in FV and DV. This association establishes how architecture decisions are implemented using cross-references as detailed in Chapter 7. This design is specified in the functional and deployment views expressed as models instead of source code.

9.3.1 Task 3.1 - Design FV

IT architects design the functional view (FV) using a component&connector model to detail the data flow in BDA pipelines and incorporating analytics models (PMML files) to specify the behavior of analytics components. Reviewing the ACCRA patterns catalog, this solution could be implemented as a big data pipeline to ingest, predict, and export churn predictions. This pipeline is composed of a data collector, an analytics engine, and a data sink following the patterns catalog. To integrate these components, the ACCRA catalog offers the distributed message broker, which decouples producer and consumers in an event-oriented connection.

The FV specification of the running example using the FV metamodel and the DSL is detailed in Listing 9.4. This *ChurnFVModel uses* (line 2) the architecture drivers previously defined in the *ChurnProject* model (see Section 9.1.2). This cross-reference enables us to associate the architecture decision *D1* to components and connectors. The FV includes the *Collector* component (lines 4-12) to expose a HTTP endpoint which receives the HTTP request with a customer data input in JSON format, and a *Writer* component (lines 22-27) to export the churn estimation to MongoDB. The *ChurnEstimator* component (lines 13-21) receives data from the *Collector* and delivers its estimation to the *Writer* component. The *ChurnEstimator* defines its analytics behavior through the PMML file (line 15) that was created in the previous activity. All these components use a microbatch processing model and exactly-once delivery guarantee. Finally, *InputQueue* (lines 30-38) and *OutputQueue* (line 39) connect the components in an event-driven communication with exactly-once delivery guarantee.

Listing 9.4: Extract of the FV specification

```
1  FunctionalView ChurnFVModel
2  use project ChurnProject{
3          Components {
4                  Ingestor Collector {
5                          type : HTTP
6                          procModel:MICROBATCH delivery: EXACTLY_ONCE
7                          format:"JSON" conn:"<ENDPOINT>"
8                          ports: {
9                          Port custData:PROVIDED
10                                 fields: {"VMailMessage":Int, ..., "State":String}
11                         }
12                 },
13                 Estimator ChurnEstimator {
14                     procModel:MICROBATCH delivery: EXACTLY_ONCE
15                     pmml:"file:///ChurnModel.pmml"
16                     decision : D1
17                     ports:{
18                     Port custDataIn:REQUIRED
19                         fields: {"VMailMessage":Int, ..., "State":String},
20                     Port churnOut:PROVIDED ...}
21                 },
22                 Sink Writer {
23                     type:DATABASE
24                     procModel:MICROBATCH delivery: EXACTLY_ONCE
25                     conn:"mongodb://<USER>:<PASS>@<HOST>:27017/churn"
26                     ports: {Port sinkIn:REQUIRED ...}
27                 }
28         }
29         Connectors {
30                 Event InputQueue {
31                 delivery: EXACTLY_ONCE
32                 buffering: BUFFERED
```

```
33                  props:"topic:custData"
34                  roles : {
35                  Role reqQIn:IN  -> custData,
36                  Role reqQOut:OUT -> custDataIn
37                      }
38                  }
39                  Event OutputQueue {...} ...
40      }
41  }
```

9.3.2 Task 3.2 - Design DV

IT architects specify the deployment view (DV) by defining technology infrastructure. In addition, DV includes software artifacts mapped to the infrastructure where they will be installed and executed. The architect assigns a specific technology to each artifact to instantiate the technology-neutral models to a concrete software solution. In addition, the architecture decisions and tactics related to increasing available resources and parallel processing defined as architecture drivers are materialized in this view by defining replicas to artifacts.

Listing 9.5 details an DV extract of the running example. The DV model *ChurnDVModel* uses the *ChurnProject* (line 2) and *ChurnFVModel* (line 3) to import the architecture drivers, components and connectors. Each component and connector from FV must be defined as an *Artifact* assigned to a concrete technology to be deployed. In our example, *ChurnEstimator* (lines 5-8) will be deployed using Apache Spark and event connectors using Apache Kafka, like *ReqQArt* in lines 9-12. The devices which constitute the underlying infrastructure are defined from line 14 to 19. These devices can then be mapped to pods or using the label selectors implemented in Kubernetes. The Apache Spark and Kafka clusters are provisioned according to the deployments in lines 20-36. A deployment example of the Spark workers with three replicas is shown in lines 21-35, assigning the deployed artifacts (*ChurnArt* in line 30), the container image (line 32), and resource requirements (line 33).

Listing 9.5: Extract of the DV specification

```
1   DeploymentView ChurnDVModel
2   use project ChurnProject
3   use functionalView ChurnFVModel {
4           Artifacts{
5                   Artifact ChurnArt {
6                           component: ChurnEstimator
7                           technology: Spark
8                   },
9                   Artifact InputQArt{
10                          connector: InputQueue
```

```
11                          technology: Kafka
12                  },...
13          }
14          Devices{
15                  Device a {
16                          host: "a" type:MEDIUM
17                          cpu: 2 storage: 100 memory: 8
18                  },...
19          }
20          Deployments{
21                  Deployment SparkWorkerDep {
22                          replicas: 3
23                          decision: D1
24                          pods {
25                                  Pod SparkWPod{
26                                          device: a
27                                          Envs{
28                                            ExecEnv SparkWEnv{
29                                              deployedArtifacts{
30                                                ChurnArt, ...
31                                              }
32                                              image: "ramhiser/spark:2.0.1"
33                                              cpu_req: 0.3 mem_req: 1000
34                                              ...
35                                            }...
36  }
```

9.4 Activity 4 - Validation

Once FV and DV models are defined, they are validated against architecture constraints as presented in Chapter 8. The ACCORDANT Validator checks the default consistency, completeness, and correctness properties, along with the project-specific constraints defined as architecture drivers. To illustrate this Activity 4 with the running example, a subset of constraints presented in Chapter 8 will be reviewed. We select the constraints subset that could be more representative for this running example, see Table 9.4.1. *IF1* and *ID1* are the specific constraints defined in the architectural drivers. *IF1* and *F11* holds because all processing models of components and connectors were defined as MICROBATCH, see lines 7, 15, and 25 in Listing 9.4. The delivery guarantee of components and connectors were defined as EXACTLY_ONCE therefore *F12* holds, see lines 7, 15, 25, and 32 in Listing 9.4. *F14* checks the connected ports structure (data type and order), and they match as detailed in lines 11 and 20 in Listing 9.4.

Regarding the ontology-supported constraints, *FD5* validates that the algorithm defined in the PMML file (DecisionTree) is supported by the technology assigned to the estimator artifact in the DV, i.e., Apache Spark. This constraint is implemented as a SPARQL query on the ACC-

Table 9.4.1: Constraints validated in the running example

Code	Constraint	Type
IF1	All components must match with processing models constraints	Inter-view
ID1	All selected technologies must match with license constraints	Inter-view
F11	All components' processing models must match	Intra-view
F12	All components and connectors delivery guarantee must match	Intra-view
F14	All connected ports' fields must be compatible	Intra-view
FD5	All Estimator algorithms must be supported by available technologies	Inter-view

OWL ontology that returns the tuple: ("Spark", "DecisionTree") as defined by the official vendor documentation[3]. For more details about the SPARQL queries, please revisit the Listing 8.3. *ID1* holds since Apache Spark and Kafka are under Apache license, and *Apache* is a subtype of *Opensource* license in the ACC-OWL ontology. Listing 9.6 details the SPARQL query for constraint *ID1* that returns tuples: {("Spark", "Apache"), ("Kafka", "Apache")}

Listing 9.6: SPARQL query for ID1

```
1 SELECT DISTINCT ?tech ?license
2 WHERE {?tech rdf:type accowl:Technology .
3 ?tech accowl:hasTechLicense ?license .
4 ?license rdf:type ?ltype .
5 ?ltype rdfs:subClassOf <licenseKind> .
6 FILTER ( ?tech = <tech> ) }
```

9.5 Activity 5 - Code generation

The ACCORDANT MF interweaves the models from multiple views and generates the functional and deployment (IaC) code as detailed in Chapter 7. This code generation traverses the FV and DV models, and it is supported by model-to-text transformations based on the target-technology assignment. Continuing with our running example, the Spark, Kafka, and Kubernetes Xtend templates generate code for Spark applications, Kafka connectors, and Kubernetes deployments.

Listing 9.7 shows an extract of the Apache Spark code generated to implement the churn estimator component. Lines 4-7 comprise the configuration and creation of Spark context and session. The churn model in PMML is loaded and built in lines 8-9. Lines 12-18 comprise the construction of the data inputs as defined in the PMML file. In lines 19 and 20, the PMML

[3]https://spark.apache.org/docs/latest/mllib-decision-tree.html

transformer predicts the churn likelihood based on the data input. The instrumentation code to log the prediction latency using Prometheus is implemented in lines 21 and 22.

Listing 9.7: Apache Spark code generated for the ChurnEstimator component

```
1  ...
2  public class ChurnEstimator {
3    public static void main(String[] args) throws Exception {
4      SparkConf conf = new SparkConf().setAppName("churEstimator")
5          .setMaster("local[*]");
6      JavaSparkContext sc = new JavaSparkContext(conf);
7      SparkSession sparkSession = new SparkSession(sc.sc());
8      InputStream pmmlFile = new URL("file:///ChurnModel.pmml").openStream();
9      EvaluatorBuilder evalBuilder = new LoadingModelEvaluatorBuilder().load(pmmlFile);
10     ...
11     Evaluator evaluator = evalBuilder.build();
12     TransformerBuilder builder = new TransformerBuilder(evaluator)
13             .withTargetCols().exploded(true);
14     List<StructField> fields = new ArrayList<StructField>();
15     fields.add(DataTypes.createStructField("VMailMessage",
16         DataTypes.IntegerType, true));
17     ...
18     StructType schema = DataTypes.createStructType(fields);
19     Transformer pmmlTransformer = builder.build();
20     Dataset<Row> resultDs = pmmlTransformer.transform(inputDs);
21     receivedBytes.observe(inputDs.size());
22     requestTimer.observeDuration();
23         ...
24     sparkSession.close();
25     sc.close();
```

Listing 9.8 details the infrastructure code generated in the ChurnDV.yml for the Kubernetes deployment of the Spark cluster. The Kubernetes deployment specifies three replicas in line 6. The container image is defined in line 10. The exported port is stated in line 13, and requested resources are specified in lines 16 and 17.

Listing 9.8: Kubernetes code generated for the Spark deployment

```
1  apiVersion: apps/v1
2  kind: Deployment
3  metadata:
4    name: SparkWorkerDep
5  spec:
6    replicas: 3
7      spec:
8        containers:
9        - name: SparkWEnv
10         image: ramhiser/spark:2.0.1
11         command: [/spark-worker]
12         ports:
13         - containerPort: 8081
```

```
14          resources:
15            requests:
16              cpu: 0.3
17              memory: 1000Mi
18  ...
```

9.6 Activity 6 - Code execution

The ACCORDANT framework provisions the infrastructure semi-automatically using Kubernetes tools to execute the code generated in the previous activity. The functional code is pushed to a git repository and then pull into the container image. The infrastructure provision is made by the architect using *kubectl*[4], the command-line tool to deploy Kubernetes YAML files. As a result, Kubernetes creates the deployment, pods, and services with the functional code cloned from the git repository into the container images. Listing 9.9 shows the deployment of the Kubernetes files and the list of the Pods deployed. The three replicas of the Spark worker are shown in the console.

Listing 9.9: Deployment of IaC code using kubectl command

```
$ kubectl apply -f ChurnDV.yml
deployment.apps/SparkWorkerDep created
service/spark-service created
$ kubectl get pods
NAME              READY       STATUS      RESTART     AGE
SparkWorkerDep    3/3         Running     0           11s
```

9.7 Activity 7 - Solution monitoring

Once the solution is running, The ACCORDANT framework can monitor metrics previously specified during the architecture design using Prometheus and Grafana. Grafana dashboard queries the metrics using PromQL (Prometheus Query Language) to visualize them against the QS mapped to the components and connectors. Fig 9.7.1 shows a screenshot of the Grafana dashboard of the churn estimator latency (green line) highlighting the QS threshold of 2 seconds (red line).

[4]https://kubernetes.io/docs/reference/kubectl/kubectl/

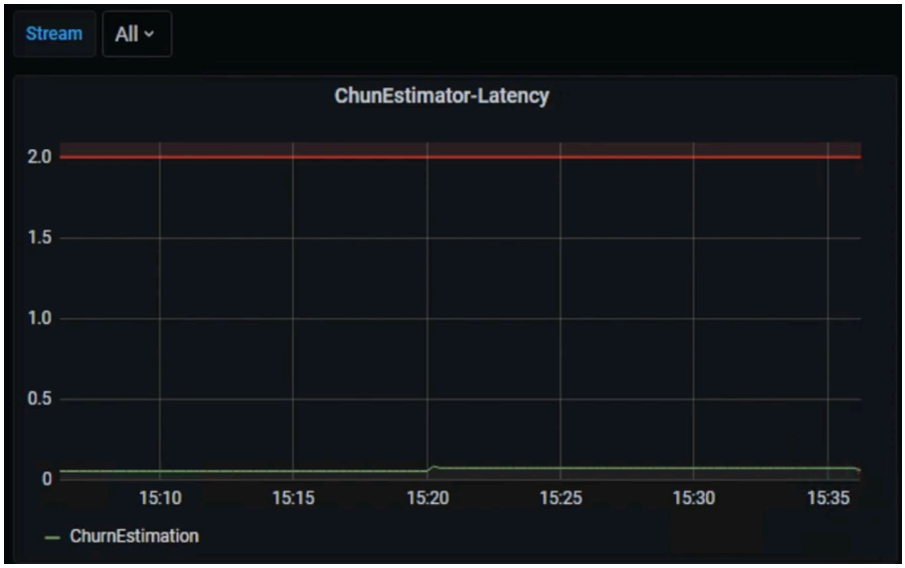

Figure 9.7.1: Quality scenario monitoring of ChurnEstimator in Grafana

10
The ACCORDANT Tool

This chapter describes the ACCORDANT Tool, a system that implements the ACCORDANT method integrating the modeling framework, architecture validator, and a web application.

10.1 The system architecture of the ACCORDANT Tool

The overall system architecture is shown in Fig. 10.1.1. In gray, we highlight the components developed within this proposal, while the third-party components used in the system are left in white. This architecture includes the following main components:

10.1.1 The ACCORDANT modeling framework (MF)

The ACCORDANT Modeling Framework (MF) presented in Chapter 7 is the core component of the system. ACCORDANT MF is built on the Eclipse Modeling Framework (EMF)[1] and uses Xtext to implement the DSLs. The ACCORDANT models are stored in a file system repository in XMI format (XML Metadata Interchange). In addition, the code generation is

[1] https://www.eclipse.org/modeling/emf

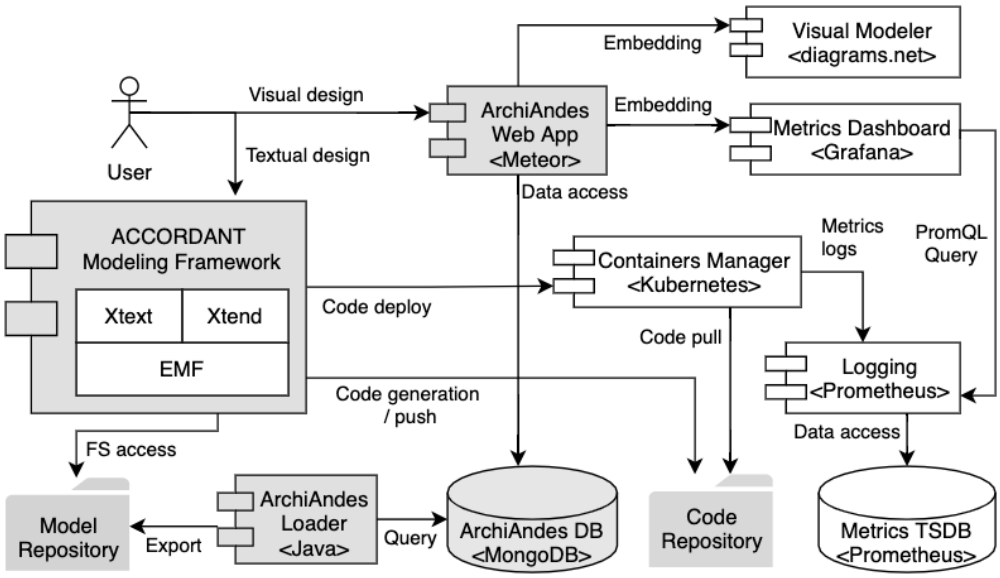

Figure 10.1.1: The ACCORDANT Tool system architecture.

implemented using the model to text transformations in Xtend. The generated software and infrastructure code is pushed into a git repository. Eclipse is the IDE to use the ACCORDANT MF. ACCORDANT plugins must be installed to use the DSLs developed in Xtext. The ACCORDANT MF Eclipse projects are available in a Github repository[2].

10.1.2 Archiandes

ArchiAndes is a web application that provides a user interface to design models that conform to ACCORDANT metamodel. ArchiAndes is developed in Meteor, and a MongoDB instance is used for persistence. This front-end component embeds a diagrams.net modeler (previously known as draw.io) [3] to facilitate the visual design of FV and DV diagrams and a Grafana dashboard to visualize the application metrics. A more detailed explanation of Archiandes will be presented in Section 10.2.

10.1.3 The visual modeler

The visual modeler embeds a diagrams.net instance that allows users to design FV and DV diagrams to promote usability. We developed ACCORDANT libraries for FV and DV to be used in diagrams.net.

[2]https://github.com/kmilo-castellanos/accordant
[3]https://www.diagrams.net

10.1.4 The ArchiAndes model loader

ArchiAndes Model Loader is a Java standalone program to export models from ArchiAndes' MongoDB database to the ACCORDANT model repository in XMI format. The Model loader receives the MongoDB connection as a parameter to connect and extract the data from ArchiAndes and the target folder to store the XMI files generated.

10.1.5 Containers Manager

We implement the containers manager with Kubernetes, given its broad adoption. The code generated by ACCORDANT MF is pulled and executed in Kubernetes containers using Kubectl. The running BDA application is instrumented to push metrics through the Prometheus client.

10.1.6 The metrics logging component

The metrics logging component is implemented in Prometheus, an open-source system monitoring and alerting toolkit. Prometheus persists application metrics into the time series database (TSBD). In addition, queries over the TSDB are executed by the Grafana dashboard using PromQL (Prometheus Query Language) to visualize QS metrics.

10.1.7 Metrics dashboard

Metrics dashboard is a Grafana[4] component that supports querying Prometheus. Grafana is a multi-platform open-source and interactive visualization web application that provides charts, graphs, and alerts connected to telemetry data sources.

10.2 ArchiAndes: A web tool to design ACCORDANT models

The learned lessons during experimentation with users show that a textual DSL could harden the design of architecture diagrams for newcomers, who need to be trained in how to use the Eclipse IDE and the ACCORDANT DSL syntax. To facilitate the use of ACCORDANT by new users, we built a web application tool called ArchiAndes[5] which allows user to design the architecture views conform to the ACCORDANT metamodel using a visual editor. Visual editors are more natural than textual DSLs for architects who usually design architecture diagrams in tools such

[4]https://grafana.com
[5]https://github.com/kmilo-castellanos/ArchiAndes

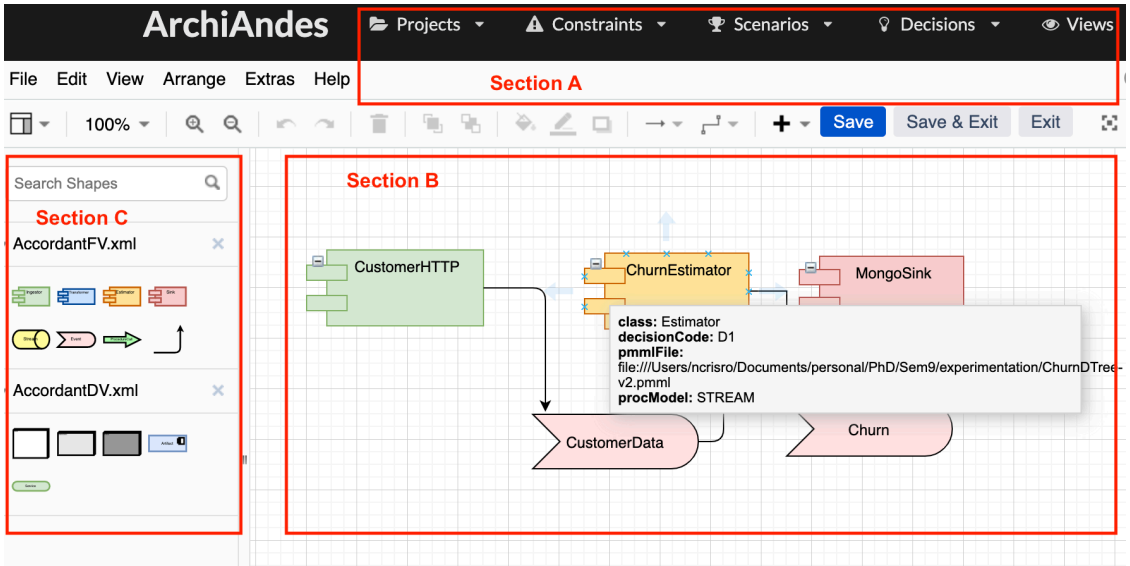

Figure 10.2.1: ACCORDANT visual modeling using ArchiAndes UI

diagrams.net[6] (formerly draw.io), Lucidchart, Visual Paradigm, among others. ArchiAndes offers a web interface to specify the same elements specified in the ACCORDANT metamodel. Section A in Fig.10.2.1 shows the menu options to manage projects, constraints, QS, decisions, and views. ArchiAndes also embeds diagrams.net editors, as shown in Section B, to specify architectural drivers and design FV and DV models. We developed libraries in XML to extend the diagramas.net with the FV and DV visual elements, as highlighted in Section C. The visual elements contain a set of attributes to define the ACCORDANT properties required by the metamodel. Once architects design the ACCORDANT models in ArchiAndes, those models can be loaded in the ACCORDANT MF to be integrated, validated, and deployed.

[6]https://www.diagrams.net

Part III

Evaluation and Conclusions

11
Evaluation

In this chapter[1], we evaluate our approach presented in Part II. This chapter aims at testing the hypotheses H1, H2, and H3 stated in Section 1.4. Our evaluation follows the DSR methodology presented in Section 1.7. Specifically, we apply *EVAL2* to evaluate artifact design to validate this design contributes to the solution of the stated problem, and *EVAL3* to initially demonstrate whether and how well the artifact works when interacting with users in an artificial setting. Table 11.0.1 summarizes the evaluation schema for each ACCORDANT component, its related hypotheses, the applied methods, and the chapter sections that cover each evaluation. We apply a case study validation because it is suited for software engineering research where the objects of study are contemporary phenomena, and they are hard to study in isolation [84].

The rest of this chapter is organized as follows. Section 11.1 details the evaluation of the reference architecture for BDA. Section 11.2 presents the evaluation of the ACCORDANT modeling framework. Section 11.3 presents the results of a survey that evaluates ACCORDANT against

[1]Portions of this chapter previously appeared in: Castellanos, C., Varela, C. A., Correal, D. (2021). "ACCORDANT: A Domain Specific-Model and DevOps Approach for Big Data Analytics Architectures". Journal of Systems and Software, 172, 110869. Portions of this chapter have been submitted to: Castellanos, B. Perez, C., Varela, C. A., Correal, D. (2021) "Architectural Mismatch in Big Data Analytics Applications: A Domain-Specific Model Approach". In Journal of Systems and Software.

Table 11.0.1: ACCORDANT Evaluation

Component	Hypothesis	Evaluation Methods	Section
Reference Architecture	$H2$	EVAL3: Case studies	11.1
Modeling Framework	$H1, H2$	EVAL3: Case studies	11.2
		EVAL3: Experimentation	11.3
		EVAL3: Survey	
Architecture Validator	$H3$	EVAL3: Case studies	11.4
		EVAL3: Experimentation	
Method	$H1, H2, H3$	EVAL3: Case studies	11.2
		EVAL3: Experimentation	11.3
		EVAL3: Survey	11.4
Tool	$H1, H2, H3$	EVAL2: Prototype	11.2
		EVAL3: Case studies	11.3
		EVAL3: Experimentation	11.4
		EVAL3: Survey	

a similar approach. Section 11.4 describes the evaluation of architectural mismatch detection in ACCORDANT. Finally, Section 11.5 summarizes the results of the chapter.

A crucial part of our experimentation has been developed within the context of the CAOBA Alliance[2] that carries out applied research. CAOBA is the Center of Excellence and Appropriation in Big Data and Data Analytics, constituted by governmental institutions, universities, cross-industry companies, and technology vendors in Colombia. CAOBA is supported by the Ministry of Information Technologies and Telecommunications of the Republic of Colombia (MinTIC) through the Colombian Administrative Department of Science, Technology, and Innovation (COLCIENCIAS). CAOBA aims to promote big data technologies and data analytics through applied research, consulting services, and product development to offer solutions around BDA technologies. In CAOBA, we worked on real-life projects in fields such as transportation and tax fraud detection. Another important part of this experimentation was developed around the avionics field in the Worldwide Computing Laboratory[3] (WCL) at Rensselaer Polytechnic Institute. The WCL research fields include flight systems, cloud computing, middleware for adaptive distributed systems, concurrent programming models and languages, and software verification.

[2]https://alianzacaoba.co
[3]https://wcl.cs.rpi.edu

11.1 Evaluating the reference architecture ACCRA

This section illustrates and validates the feasibility and applicability of ACCRA introduced in Chapter 6 with three BDA case studies and their concrete architectures. Given CAOBA's research context, we take advantage of transportation projects to apply ACCRA in this field with three case studies: CS1, CS2, and CS3. Table 11.1.1 details each case study along with tactics and patterns applied from the catalogs described in Section 6.3. CS1 is currently used by the Transit Authority of Bogotá, Colombia to analyze road accidents and their relationship with mobility and traffic tickets. The datasets used in CS2 and CS3 are open data, so they are publicly available. CS2 is a near-real-time analytical application to predict bus delays in the public transportation system of Vancouver, Canada. In the last case, CS3 addresses smart mobility analysis regarding pollution metrics collected by sensor networks in Aarhus, Denmark. The following sections will detail each case study, including concrete software architectures and their components mapped to ACCRA service zones to validate their compliance. In addition, each concrete architecture component specifies the open-source technologies used to implement it according to the technology selection view.

Table 11.1.1: Case Studies with applied tactics and patterns

Code	Description	Tactics	Patterns
CS1	Accidents Analysis	Unnecessary Data Removal, Feature selection and extraction	Data Refinery, Poly Storage
CS2	Bus Delay Prediction	Parallel processing, Shard data set across multiple servers, Reuse previous results	Lambda Architecture, Interactive Query Engine, Distributed Message Broker
CS3	Shortest Paths based on Pollution	Unnecessary Data Removal, Feature selection and extraction, Parallel processing, Shard data set across multiple servers	Data Collector, Data Refinery

11.1.1 CS1: Accidents analysis in Bogotá, Colombia

A web application was implemented to offer a descriptive and predictive analysis of accidents and the monitoring of public transportation operations in the transit authority of Bogotá. The accidents module analyzes historic accident data and their relationship with traffic tickets to

support road safety decisions. The operation monitoring module offers the public transport system's historical tracking to understand mobility patterns. These modules are integrated within a dashboard that allows users to interact with accidents history, bus trips, and dimensions such as geolocation, type, vehicle, date, and route. The predictive analysis included calculating accident risk indices by road segment and correlations of accidents and traffic tickets associated with the ticket's cause. As a result, the transit authority of Bogotá can now analyze historical data to discover insights, define safety policies supported by these data, and measure the results of these decisions.

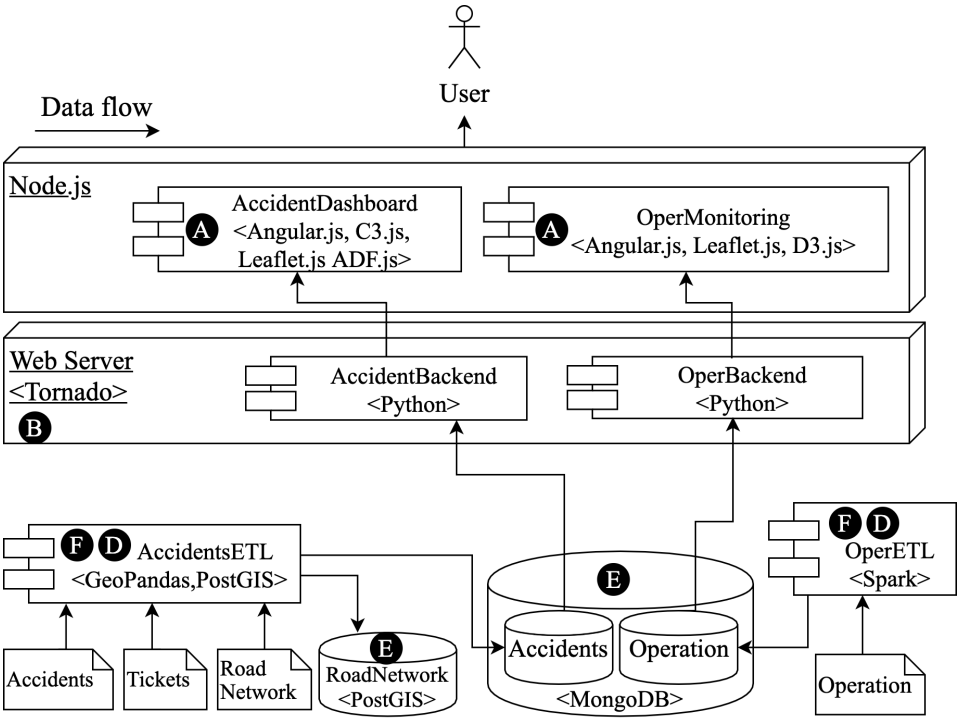

Figure 11.1.1: CS1: Accident Analysis in Bogotá

Fig. 11.1.1 describes the concrete software architecture, which was designed following ACCRA, and Table 11.1.2 details the mapping between components and ACCRA services. Accidents, traffic tickets, and road networks are the external data sources loaded by the AccidentsETL component, which is part of service zones *F* and *D* of ACCRA, and is implemented using Python, Pandas, and PostGIS. In terms of volume and variety, data were composed of structured data such as traffic tickets (1,598,199 records), accidents (2,385,000 records), bus GPS traces (263,937,822 records), and road network (137,962 segments) in a graph network.

Table 11.1.2: CS1 Components mapped to ACCRA functional view

Components	Zone	Service
AccidentsETL	F, D	Ingestion, Cleaning, Transformation
OperETL	F, D	Ingestion, Cleaning, Transformation
Accidents, Operations	E	Document Store
RoadNetwork	E	Spatial DB
AccidentBackend	B, C	Descriptive Analysis in Road Safety
OperBackend	B, C	Descriptive Analysis in Traffic Management and Ops.
Accident and Operation UI	A	Dashboards Visualization

These data sources are cleaned, filtered, integrated, and stored in a batch in a MongoDB database once a new dataset arrives; these steps correspond to service zone *E*. Besides, historic bus operation data are loaded and integrated by the component *OperETL* implemented in Spark in zones *F* and *D*. The resulting aggregated operational data is also stored in MongoDB. In the zone *B*, the components *AccidentBackend* and *OperBackend* access and aggregate pre-processed data of accident and operations using Python, and then expose analysis services via REST consumed by frontend components. Finally, frontend components *AccidentDashboard* and *OperMonitoring* are built within a dashboard using technologies such as Angular Dashboard Framework (ADF), AngularJS, C3, D3, and Leaflet within service zone *A*.

11.1.2 CS2: Bus delay prediction in Vancouver, Canada

Case Study 2 implements a Lambda architecture to analyze near real-time data of bus trips in the public transport system of Vancouver[4]. A detailed description of this case study implemented in a Software as a Service model can be found in [76]. Trip updates are collected and aggregated by bus route to calculate the delay average per time window, five minutes. Data processing has low-latency analytical constraints and limited accuracy due to the restriction to micro-batches of data. On the other hand, batch processing allows to build robust analytical models, it does not offer adequate response times. This application allows users to accurately predict the bus delays to plan their trips and the transport authority to manage their operation better.

A Lambda architecture was designed to be compliant with ACCRA as detailed in Fig. 11.1.2, and Table 11.1.3 details the mapping between components and ACCRA services. The Vancou-

[4]https://developer.translink.ca

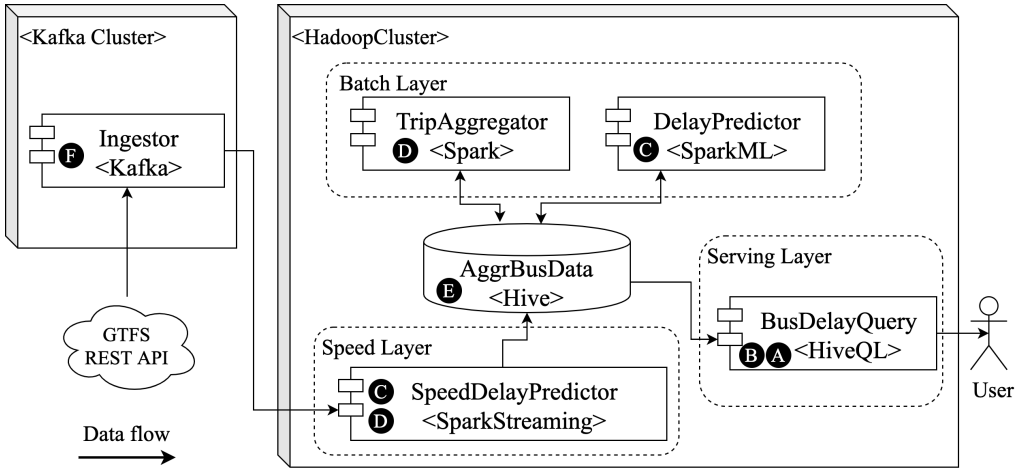

Figure 11.1.2: CS2: Delay prediction at Vancouver

Table 11.1.3: CS2 Components mapped to ACCRA functional view

Components	Zone	Service
Ingestor	F	Ingestion
SpeedDelayPredictor	D, C	Transformation, Predictive Analysis in Traveler Information
TripAggregator	D	Transformation
DelayPredictor	C	Predictive Analysis on Traveler Information
AggrBusData	E	OLAP
BusDelayQuery	B, A	Traveler Information, Interactive Analytics Tools

ver real-time travel update data were delivered in General Transit Feed Specification (GTFS) format. Each GTFS trip update is generated every 60 seconds and processed in the pipeline downstream. Each trip update is a binary protobuffer file (56KB on average), which has to be deserialized to JSON format. Trip update feeds are consumed by an ingestion component using Apache Kafka (service zone *F*) to provide low latency and high availability. In the speed layer (zone *C*), the delay predictor component consumes data from the Kafka topic to aggregate and predict the delay expected per time window using Spark Streaming and storing the speed views in HDFS. Besides, each raw trip update is stored in HDFS to be further processed by the batch layer. The batch layer was implemented using Spark and SparkML Lib, where TripAggregator component (zone *D*) processes historical data to be stored in HDFS. DelayPredictor (zone *C*) estimates the expected delay per time window and bus route, storing the outputs as batch views in HDFS. In the serving layer (zones *B* and *A*), both speed and batch views are integrated using interactive queries by the users through a query component using HiveQL.

11.1.3 CS3: Smart mobility in Aarhus, Denmark

Smart mobility is a core concept in smart city strategies, and it supports the management of traffic jams in cities with increasingly high urbanization rates. To illustrate our approach generalization to other smart mobility case studies, we align ACCRA to the architecture presented in [104], which analyzes road traffic and pollution using sensor data. The datasets include pollution, road traffic, weather, and parking. Road traffic and pollution were collected every five minutes from semi-structured datasets in CSV format from the Aarhus' Open Data. A graph is built with the sensors network dataset to calculate the shortest paths between locations. Then, the least polluted paths are calculated based on the pre-calculated shortest paths and pollution data collected by sensors network using a Hadoop cluster. This application enables final users to select healthier paths and transport authorities to monitor the environmental variables to improve informed decision-making.

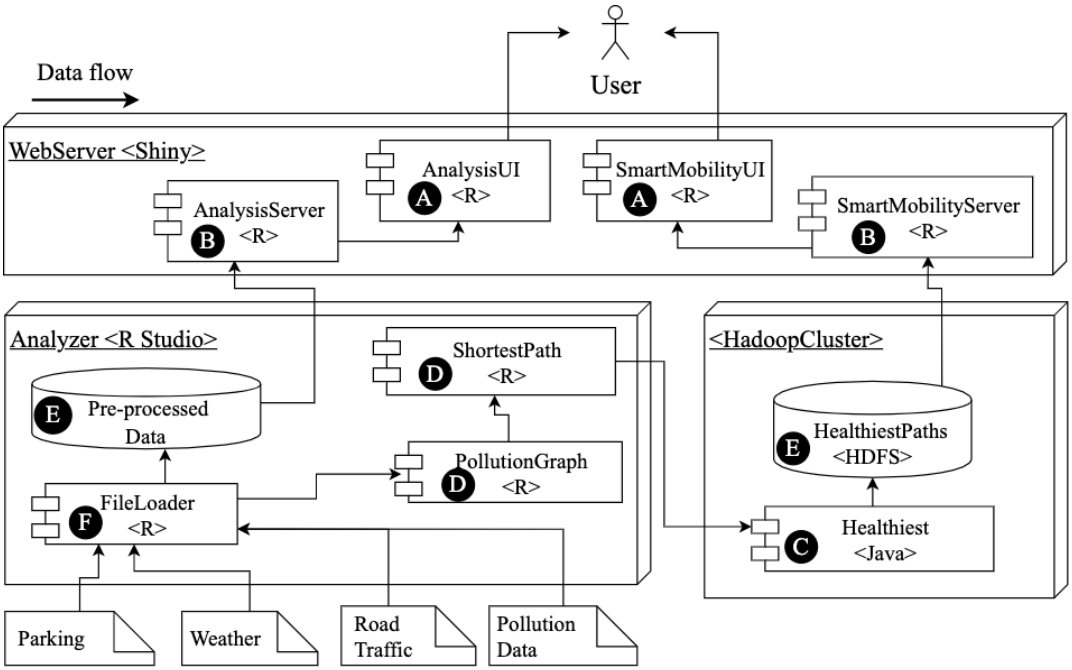

Figure 11.1.3: CS3: Smart Mobility in ACCRA

Fig. 11.1.3 details the software architecture CS3, as an adaptation of the one presented in [104], and Table 11.1.4 details the mapping between components and ACCRA services. Data sets of parking, weather, road traffic, and pollution are ingested and pre-processed by a file loader implemented in R (zone F). These pre-processed data are taken by two components (zone D)

Table 11.1.4: CS3 components mapped to ACCRA functional view

Components	Zone	Service
FileLoader	F	Ingestion
PollutionGraph	D	Cleaning, Transformation
ShortestPath	D	Transformation
Healthiest	C	Descriptive Analysis
HealthiestPaths, Pre-processed Data	E	Distributed File System
AnalysisServer, SmartMobility	B	Traveler Information, Environmental Monitoring
AnalysisUI, SmartMobilityUI	A	Dashboard Visualization

to build a graph network and calculate the shortest paths in HDFS (zone *E*). Hadoop processes these shortest paths to calculate the healthiest path between two points (zone *C*). In Shiny, backend components *AnalysisServer* and *SmartMobilityServer* consume the pre-processed and healthiest paths data to be delivered to UI components. These backend components are located in the backend services (zone *B*) of ACCRA. Regarding presentation zone *A*, *AnalysisUI*, and *MobilityUI* components provide a dashboard using a Shiny[5] server.

11.1.4 Discussion

BDA projects present particular challenges, such as communicating with business users to identify objectives, selecting various modern tools, and using analytical techniques that require specific knowledge. These challenges imply steep learning curves that delay the construction of these types of solutions. In this work, RA proposals were reviewed, summarizing the concerns and expected capacities in BDA common to different authors. Although previous works have proposed reference architectures for ITS and BDA separately, we argue that those works lack of detailed guidance on implementation, integration, technologies, patterns, and tactics catalogs to support the development of these projects in practice. Our thesis aims to tackle these challenges by offering an RA, technology selection, and pattern catalogs to guide and facilitate these projects and their application using three case studies.

Our case studies have shown that ACCRA covers various BDA applications in the transportation by instantiating concrete architectures, patterns, and tactics. Although the technology selection view is not exhaustive, it has been sufficient to support all technologies used in these case studies. However, other projects can also require different open-source technologies. Hence both technologies and catalogs are incremental artifacts.

[5] https://shiny.rstudio.com/

Table 11.2.1: Use Cases

Use Case	Description	Domain	Analytics Model	Processing Model	QS Metric
UC1	Transport Delay Prediction	Transportation	Regression Tree	Stream	Update time, Latency
UC2	NMAC Risk Analysis	Avionics	K-means	Batch	Deadline
UC3	NMAC Detection	Avionics	Decision Tree	Micro-batch	Latency
UC4	El Niño/Southern Oscillation	Weather	Polynomial Regression	Batch	Deadline

11.2 Evaluating ACCORDANT MF for BDA development and deployment

Our experimentation aims to compare the development and deployment time for each iteration using ACCORDANT and other two frameworks reviewed in Chapter 4: FastScore and SpringXD. We chose these frameworks because they are the closest to our approach, and they support portable analytics models (PMML or PFA). We validated our work in different domains through four use cases: UC1) Transport delay prediction, UC2) Near mid-air collision (NMAC) risk analysis, UC3) Near mid-air collision detection, and UC4) El Niño/Southern Oscillation cycles. Table 11.2.1 summarizes the use cases, domains, processing models, and quality attributes. These use cases are applied to analytics models. They also illustrate BDA facets as streaming and micro-batch to deal with the velocity aspect, and batch processing is focused on volume in terms of data size and computation complexity. Fig. 11.2.1 details the component-connector model for each use case to illustrate the functional building blocks and their composition as BDA pipelines. The ACCORDANT specification of these use cases is publicly available[6], and the use cases description will be presented below.

Two development teams were trained to develop and deploy the use cases using the three approaches to compare the development time. Each team tracks the time spent in each iteration. Each development team followed the ACCORDANT method detailed previously in Chapter 9 to design, develop, and deploy the four use cases. For each use case, a set of architecture drivers, FV, and DV models were designed. After the code generation and deployment, each team evaluates the performance behavior and iterates until the solution achieves the expected QS. The

[6]http://github.com/kmilo-castellanos/accordant-usecases

resulting models for each use case are also available in the same public repository.

11.2.1 Use case 1 (UC1)

The first use case was presented in [16], and it deals with delay prediction of public transportation in Vancouver. Bus trip data is collected in real-time from Vancouver Transport Operator, and it contains bus stops, routes, and time. A regression tree model to predict bus delays (in seconds) is built, evaluated, and exported to PMML. The pipeline, described in Fig. 11.2.1a, starts with an ingestor component that receives an HTTP request and puts it into an event connector (message broker), then the request message is consumed by the estimator to predict the delay time, and queue it, to be stored into a No-SQL database (hierarchical). The PMML model is deployed into a production environment as a delay predictor service, using OpenScoring, Kafka message broker, and MongoDB writer as target technologies. The QS was defined in terms of performance and modifiability attributes. The QS specifies that users make 1000 requests to the delay prediction service under operations without load, and the responses must have an average latency lower than 2 seconds. The second QS states that when a data scientist produces a new version of the predictive model (new PMML file), it must be updated at runtime within 10 seconds.

11.2.2 Use case 2 (UC2)

UC2 was applied in aviation safety to detect near mid-air collisions (NMAC) on different air space ranges with different deployment models while performance QS is monitored. This use case is described in Fig. 11.2.1b), and it was presented in [17]. NMAC detection comprises a pairwise comparison of flights: (C_n^2), where n is the number of flights. Each comparison requires calculating distance and time based on location, speed, and heading to determine the risk level of NMAC, which implies an intensive computation of quadratic time complexity. Eight hours of data were stored in a distributed file system to be loaded by the JSON reader component. This ingestor calls the NMAC detector, which computes the alert level. Once an alerting level is calculated for each flight pair, the results are sent to the clustering estimator to be associated with a specific cluster. NMACs are stored back in the file system. To compare different data size magnitudes, we collected flight data for four air space ranges in nautical miles (nmi): 2 nmi, 20 nmi, 200 nmi, and 1,500 nmi around John F. Kennedy Airport. These ranges represent different application scopes to attend various demand levels: local, metropolitan, and regional areas. The largest dataset (1,500 nmi) is 1.4 GB of JSON files. This use case did not have real-time require-

ments due to its heavy workload nature, and therefore a performance QS for deadlines lower than one hour was defined.

11.2.3 Use case 3 (UC3)

UC3 is a real-time application to detect NMAC within an air space range, and its architecture is described in Fig. 11.2.1c). The ingestor component consumed data through the REST service. Flight data was pushed in a message queue to be consumed by the NMAC detector component, which performed the potential collision detection to be finally stored in a relational DB through a message broker connector. It is worth mentioning that the NMAC estimator of UC2 and UC3 is the same since its inputs, outputs, and behavior are identical so that we can reuse such functional component definition although its deployment can be different regarding the QS constraints. Given the near real-time nature of this application, latency is the critical quality attribute, and we evaluated this metric in two ranges of air space around John F. Kennedy Airport: 2 nmi and 200 nmi, which demand different computation resources.

11.2.4 Use case 4 (UC4)

In this last use case, we used a publicly available data and PMML model (polynomial regression) of El Nino/Southern Oscillation (ENSO)[7] to implement a batch-oriented pipeline, see Fig. 11.2.1d. The El Niño/Southern Oscillation (ENSO) cycle was the strongest of the century, which produced many problems worldwide, affecting South and North American countries with destructive flooding in some areas and intense drought in other areas. The data used in this use case is generated by a series of buoys positioned throughout the equatorial Pacific. The data contains oceanographic and surface meteorological readings: geolocation, humidity, surface winds, sea surface temperatures, and subsurface temperatures. These data help with the understanding and prediction of ENSO cycles. We read the historic data from 1980 to 1998 (178,080 records) using a CSV reader (ingestor) component, which sends the data to the ENSO predictor component. ENSO predictor is an estimator component that forecasts air temperature and stores the prediction in a distributed file system. The QS defined for UC4 was a deadline for batch processing lower than 30 minutes.

[7] http://dmg.org/pmml/pmml_examples

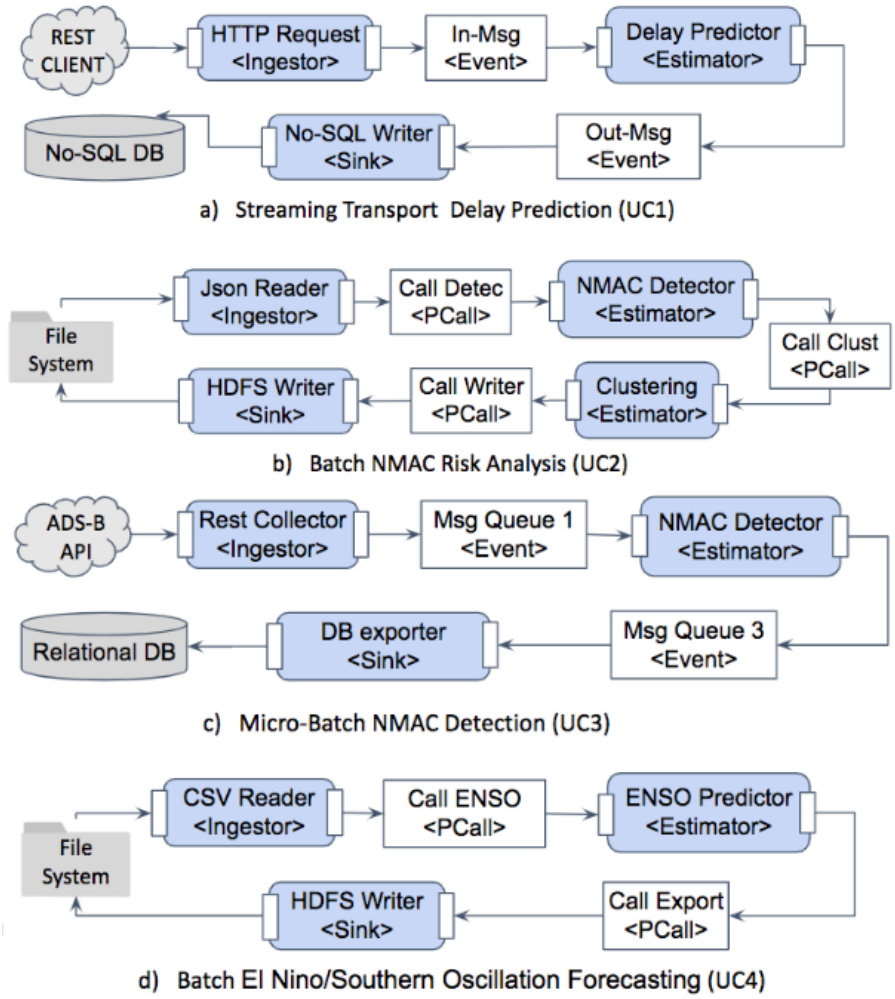

Figure 11.2.1: Component diagrams of Use Cases

11.2.5 Development, deployment time, and gain factor

To compare ACCORDANT, SpringXD, and FastScore, we measured the time invested in development and deployment phases for each use case. *Development phase* involves the design and development of the functional components and connectors in a specific technology. *Deployment phase* comprises the design and provisioning of the technology infrastructure, the installation of software artifacts developed in the previous phase, and the monitoring of the solution regarding the predefined QS. The development team performs each phase iteratively, and they can make some improvements and refinements in each iteration until achieving the expected QS. We define the gain factor as a form to measure the incremental improvement of using high-level abstractions to modify or refine an application until it achieves an expected QS. Each development team tracks the time spent in each iteration. Therefore, we measure the time invested in each iteration I, and we also calculate the gain factor $GF(uc, f)$, as a metric to estimate the cumulative average of time reduction ratio for a use case uc, and framework f. $GF(uc, f)$ is defined as follows:

$$GF(uc, f) = \frac{1}{I} \sum_{i=1}^{I} \frac{time\_spent(uc, f)_i - time\_spent(uc, f)_{i+1}}{time\_spent(uc, f)_i} \tag{11.1}$$

11.2.6 Experimentation with use cases

To design, develop, and deploy the four use cases, we followed the ACCORDANT method detailed previously in Fig. 5.2.1. We developed a domain-specific language (DSL) for each ACCORDANT metamodel presented in Section 7.2: architectural drivers, FV, and DV. These DSLs allow architects to design each use case's model following the ACCORDANT method. For the sake of brevity, this section details the step-by-step implementation of UC4 as an example, more details about the other use cases can be found in [16, 17].

11.2.6.1 Definition of architectural drivers

QS is defined regarding the use case's quality requirements. In UC4, a scheduled job to estimate ENSO cycles for ten years of data is processed in batch. In this vein, Fig 11.2.2 details architectural drivers of UC4 expressed using the respective DSL. The predictor component is required to have a deadline lower than 1 hour in the QS *UC4_QS1* (line 8). Analyzing this QS, the decision (*UC4_SP1*) is made (lines 15-18) to achieve the deadline metric by applying two tactics: *introduce concurrency* and *increase available resources*. These tactics will be materialized in the software

```
1  Project UC4{
2      QScenarios {
3          QS UC4_QS1 {
4              QA: PERFORMANCE
5              stimulus: "Scheduled job to estimate ENSO cycles for 10 years of data in batch processing"
6              environment: "Normal mode"
7              response: "The system must estimate the ENSO cycles less than one hour"
8              measure: DEADLINE between 55.0 and 60.0 in MINUTES
9          }
10     }
11     analyzedQScenarios{
12         AQS UC4_AQS1 of UC4_QS1 {
13             reasoning :"This QS requires to estimate the air temperature over an immutable large datas
14             ArchDecisions{
15                 Decision UC4_SP1{
16                     rationale: "To introduce concurrent/parallel processing can reduce the processing
17                     code: "UC4-SP1"
18                     tactics     (IntroduceConcurrency,IncreaseAvailableResources)
19                 }
20             }
21         }
22     }
23     tactics{
24         Tactic IntroduceConcurrency {
25             QA: PERFORMANCE
26             rationale: "If requests can be processed in parallel, the blocked time can be reduced. Con
27             stimulus: "Events arrive"
28             response: "Response generated within time constraints"
29             viewpoint: "Functional"
```

Figure 11.2.2: Excerpt of drivers models of UC4 using ACCORDANT DSLs

architecture design.

11.2.6.2 Development of data transformations and analytics model

Analytics Model is trained and evaluated by the data scientist outside the ACCORDANT framework, and the resulting models were exported to the PMML file to be loaded in the ACCORDANT functional model. In this case, the polynomial regression model of ENSO is downloaded and used. Fig 11.2.3 describes the structure of the PMML, detailing some data fields, mining fields, and regression coefficients. This PMML file will be embedded in the functional model in the next step.

11.2.6.3 Design of software architecture - Functional view

FV models were designed using the ACCORDANT functional DSL to specify a component-connector structure for each use case. Two iterations of the functional model were designed for UC4, and the last iteration is depicted in Fig. 11.2.4a. Since architectural drivers are required in this design, this project is imported using the keyword *use project* in line 2. The functional model specifies three components: (*CSVReader::Ingestor*, *ENSOPredictor::Estimator*, and *HDFSWriter*::Sink), and two connectors: procedure calls *CallEnso::ProcCall* and *CallEx-*

```
<PMML version="4.0" xmlns="http://www.dmg.org/PMML-4_0">
  <DataDictionary numberOfFields="7">
    <DataField name="latitude" optype="continuous" dataType="double">
      <Interval closure="closedClosed" leftMargin="-8.28" rightMargin="8.97"/>
    </DataField>
    <DataField name="longitude" optype="continuous" dataType="double">
      <Interval closure="closedClosed" leftMargin="-179.97" rightMargin="179.8"/>
    </DataField>
    <DataField name="zon_winds" optype="continuous" dataType="double">
      <Interval closure="closedClosed" leftMargin="-8.9" rightMargin="7.0"/>
    </DataField>
  </DataDictionary>
  <RegressionModel algorithmName="PolynomialRegression" targetFieldName="airtemp">
    <MiningSchema>
      <MiningField name="latitude" invalidValueTreatment="asIs"/>
      <MiningField name="longitude" invalidValueTreatment="asIs"/>
      <MiningField name="zon_winds" invalidValueTreatment="asIs"/>
    </MiningSchema>
    <RegressionTable intercept="18.475580183789134">
      <NumericPredictor name="latitude" coefficient="0.002559036916295554"/>
      <NumericPredictor name="latitude" exponent="2" coefficient="-0.001046844036920902"/>
      <NumericPredictor name="longitude" coefficient="1.6976783446054355E-4"/>
      <NumericPredictor name="longitude" exponent="2" coefficient="9.03098036941774E-6"/>
    </RegressionTable>
  </RegressionModel>
</PMML>
```

Figure 11.2.3: Excerpt of ENSO polynomial regression model of UC4 in PMML format

port::ProcCall which connect the components through ports. The components also include some properties such as connections and formats. Additionally, *ENSOPredictor* uses a batch processing model associated with the PMML "ElNinoPolReg.pmml" (line 10), obtained in the previous step, to provide the predictive behavior. The decision *UC4_SP1* aligns the architectural drivers to the ENSOPredictor in line 11. It means that *ENSOPredictor* becomes part of the *introduce concurrency* tactic realization that will be translated into a distributed processing model, which has to be supported by the target technology.

11.2.6.4 Design of software architecture - Deployment view

The deployment view models were designed using ACCORDANT DSL for each use case defined in the functional models. The UC4 deployment model had three iterations, and Fig. 11.2.4b details the last version. Given that DV is based on drivers and functional view, they are imported using keyword *use project functionalView* in lines 2 and 3, respectively. This view includes the artifacts that map connectors and components from functional view (e.g., *ENSOPredictor* in line 6) to deployable elements (e.g., *ENSOArtifact* in line 34). Each artifact is assigned to a target technology, for instance *ENSOArtifact* to Apache Spark, line 7. Devices and deployments were specified to support the computation requirements. For instance, the deployments of Spark master and worker nodes (e.g., *SparkWorkerDep* in line 27) detail the number of replicates, pods, and execution environments (*ExecEnv*). *ExecEnv* defines the docker image,

```
1  FunctionalView UC4FV
2  use project UC4{
3    Components {
4          Ingestor CSVReader {
5            type: HDFS  format:"CSV"
6        procModel:BATCH conn:"hdfs://data/enso.csv"
7        ports: { Port csv:PROVIDED}
8      },
9    Estimator ENSOPredictor {
10       procModel:BATCH  pmml:"file:////ElNinoPolReg.pmml"
11       decision: UC4_SP1
12       ports:{Port enso_in:REQUIRED, enso_out:PROVIDED}
13     },
14     Sink HDFSWriter {
15           type: HDFS  format:"CSV"
16           procModel:BATCH conn:"hdfs://data/enso-output/"
17       ports: {Port wri_in:REQUIRED}
18     }
19   }
20   Connectors {
21     ProcCall CallEnso {
22       roles : {
23         Role ce_src:IN  -> csv,
24         Role ce_dst:OUT -> enso_in
25       }
26     }
27     ,
28     ProcCall CallExport {
29       roles : {
30         Role cw_src:IN  -> enso_out,
31         Role cw_dst:OUT -> wri_in
32       }
33     }
34   }
```

```
1  DeploymentView UC4DV
2  use project UC4
3  use functionalView UC4FV {
4      artifacts{
5          Artifact ENSOArtifact {
6              component : ENSOPredictor
7              technology: Spark
8          }
9      }
10     devs{
11         Device a {
12             host: "a" type:MEDIUM
13             cpu: 2 storage: 100 memory: 8
14
15         }, …
26     deployments{
27         Deployment SparkWorkerDep {
28             replicas: 3
29             pods {
30                 Pod SparkWPod{
31                     envs{
32                         ExecEnv SparkWEnv{
33                             deployedArtifacts{
34                                 ENSOArtifact
35                             }
36                             image: "ramhiser/spark:2.0.1'
37                             cpu_req: 0.3
38                             ports [8081]
39                             commands ["/spark-worker"]
40                         }
41                     }
42                 }
43             }
44             decision : UC4_SP1
```

Figure 11.2.4: Excerpt of FV and DV models of UC4 using ACCORDANT DSLs

CPU and memory requirements, ports, and commands (lines 32-40) along with the artifacts to be deployed (*ENSOArtifact*). Finally, the decision *UC4_SP1* associates the deployment *SparkWorkerDep* in line 44. This mapping links deployments to QS indirectly, along with tactic *increase available resources* to support distributed computing over a Spark cluster.

11.2.6.5 Integration and code generation

Once the FV and DV models were designed and integrated, the code generation produced both the functional code and IaC. On the one hand, the functional code is a Spark driver program as detailed in Listing 11.1, where *ENSOPredictor* component implements the PMML model in lines 2-4. The Spark program defines data input and output from the Data Dictionary and Mining Schema embedded in PMML specifications in lines 8-12. On the other hand, infrastructure code is the configuration files that specify the provision and configuration policies of the Kubernetes cluster. Listing 11.2 shows an example of generated Kubernetes files. The whole code of use cases is publicly available in the ACCORDANT use cases repository.

Listing 11.1: Generated Java Code of EnsoEstimator Component for Spark Streaming

```
1  SparkSession sparkSession = new SparkSession(sc.sc());
2  InputStream pmmlFile = new URL("file:////path/ElNinoPolReg.pmml")
3  EvaluatorBuilder builder = new LoadingModelEvaluatorBuilder().load(pmmlFile);
4  Evaluator eval = builder.build();
5  TransformerBuilder pmmlBuilder =
```

```
6     new TransformerBuilder(eval).withTargetCols().exploded(true);
7   List<StructField> fields = new ArrayList<StructField>();
8   fields.add(DataTypes.createStructField("latitude", DataTypes.DoubleType, true));
9   ...
10  fields.add(DataTypes.createStructField("s_s_temp",DataTypes.DoubleType, true));
11  StructType schema = DataTypes.createStructType(fields);
12  Transformer pmmlTransformer = pmmlBuilder.build();
13  Dataset<Row> inputDs = sparkSession.read().schema(schema).csv("data/Elnino.csv");
14  Dataset<Row> resultDs = pmmlTransformer.transform(inputDs);
15  receivedBytes.observe(inputDs.size());
16  requestTimer.observeDuration();
17  resultDs.write().option("header", "true").csv("/enso-output/");
18  ...
```

Listing 11.2: Generated YAML Code from Deployment Specification for Kubernetes (Extract)

```
1
2   apiVersion: apps/v1
3   kind: Deployment
4   metadata:
5     name: SparkWorkerDep
6   spec:
7     replicas: 3
8       spec:
9         containers:
10        - name: SparkWEnv
11          image: ramhiser/spark:2.0.1
12          command: [/spark-worker]
13          ports:
14          - containerPort: 8081
15          resources:
16            requests:
17              cpu: 0.3
18  ...
```

11.2.6.6 Code execution

Kubernetes code was executed on the AWS cloud using Amazon Elastic Container Service for Kubernetes (Amazon EKS) and Elastic Compute Cloud (EC2). After that, the software code was installed over the cluster to operationalize the end-to-end solution.

11.2.6.7 Solution monitoring

Performance metrics for each use case in operation were collected and validated against QS defined in Section 11.2.6.1. As a result, different deployment configurations were designed, deployed, and monitored in each iteration to observe the fulfillment of QS.

11.2.7 Results

This section presents and discusses the experimental results obtained during the iterative development and deployment phases of UC1, UC2, UC3, and UC4.

11.2.7.1 Development and deployment time

Fig. 11.2.5 depicts the development and deployment time (in hours) accumulated for all iterations per use case. It is worth noting that development time using ACCORDANT is higher (between 23% and 47%) compared to SpringXD and Fastscore, but the deployment time is significantly lower (between 50% and 81%) using ACCORDANT. The higher development time can be explained by the time required in ACCORDANT to specify architectural drivers and many details in the FV. In addition, the current version of the ACCORDANT prototype generates functional code for estimators, but ingestor, sinks, and connectors still require manual. Although ACCORDANT required more effort in the development phase, this effort is rewarded during the deployment phase, where infrastructure and QS-monitoring are provided automatically aligned to drivers and FV, unlike other approaches. This benefit can be observed on the deployment time across all use cases using ACCORDANT because they are more similar than the other approaches.

The most significant time differences arise from UC2 that demands more time because it includes a more complex pipeline involving two estimators: NMAC detector and K-means clustering. Another interesting finding was that the high-level reuse of previous architectural decisions (tactics) reduced the development time, as shown by the marked decrease between use cases. This time reduction is also evidenced by the increase of gain factor among iterations detailed in Fig. 11.2.5. These results suggest that ACCORDANT is most suitable for applications involving multiple iterations or subsequent applications where reusing architectural decisions, models, and metrics can reduce development times.

11.2.7.2 Gain factor comparison

The gain factor metric presented in Section 11.2.5 was calculated for each use case and iteration of development and deployment phases as depicted in Fig. 11.2.6. ACCORDANT's gain factor was higher for all use cases in the development phase (Fig. 11.2.6a), which suggests that the high-level abstractions promote the highest reduction of development time among consecutive iterations. The highest gain factor was 0.46 in the UC3, which means reducing 46% the devel-

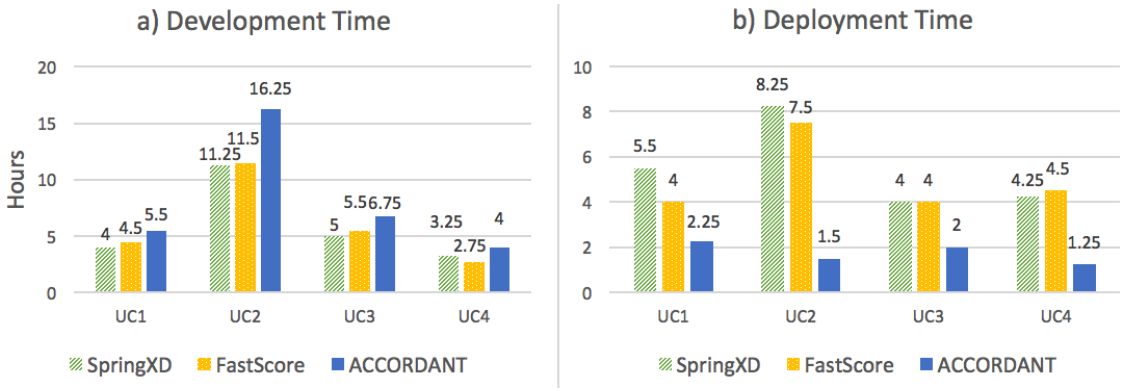

Figure 11.2.5: Development and deployment time per use case

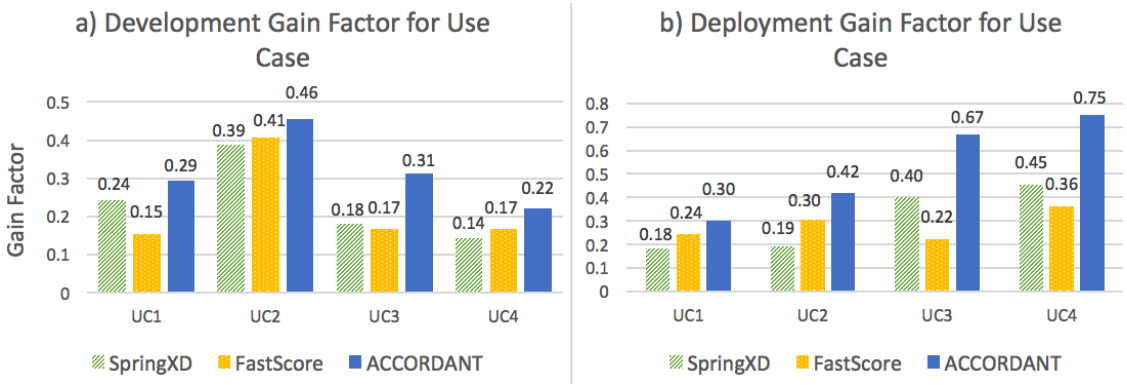

Figure 11.2.6: Gain factor per use case

opment time between consecutive iterations. The largest gain factor difference over the other approaches was 0.13 in the UC3. Regarding the deployment gain factor (Fig. 11.2.6b), ACCORDANT also exhibited the highest gain factor, up to 0.75 in UC4. This means each deployment iteration reduces the time by 75% compared to the previous one. Similar to the deployment time in the previous section, we argue that the gain factor in the deployment phase is greater because the IaC generation is not present in the other approaches.

11.3 A survey to evaluate scenarios using ACCORDANT MF

In addition to the time saving and the gain factor evaluated in Section 11.2, we also evaluate ACCORDANT on eight architecture scenarios. These scenarios cover design, analysis, and deployment tasks validated across a broader population. The case study used in the survey is a BDA application for predicting customer churn in telecommunication companies. The data used cor-

responds to the profile information, consumption, and incidents of each client. A prediction model is trained to estimate the probability that a customer cancels (churn = 1) or not (churn = 0) her/his subscription from these historical customer data. In this survey, each respondent evaluates each scenario using two tools and assigning a Likert scale score on the level of agreement or disagreement with a given statement. In this survey, we compare ACCORDANT against SCDF[8] (Spring Cloud Data Flow, formerly SpringXD) to measure architecture tasks using the case study. We select SCDF because it is the closest tool to ACCORDANT regarding the related work reviewed in Chapter 4. SCDF supports PMML and offers a visual interface to design BDA pipelines. The architecture scenarios used to evaluate both tools are:

- Scenario 1: Architectural drivers specification.
- Scenario 2: FV design.
- Scenario 3: DV design.
- Scenario 4: Deployment.
- Scenario 5: Metrics monitoring and assessment.
- Scenario 6: Architectural decision analysis.
- Scenario 7: Infrastructure scaling.
- Scenario 8: Analytics model upgrade.

11.3.1 Sample and population

The practitioners in this survey have participated in BDA projects playing roles such as project manager, business expert, requirements engineer, data scientist/analyst, data engineer software designer/developer, or IT architect. We employed *Convenience sampling* (a non-probabilistic sampling method [59]) for selecting the population because of our access to participants involved in BDA projects. Participants were available through the architecture master program offered by Universidad de Los Andes.

Inclusion and exclusion criteria enable us to choose valid answers regarding experience in BDA and consistency. This survey considered the following Inclusion criteria: (i) The respondent has industrial experience in BDA projects, and (ii) The respondent has academic experi-

[8] https://spring.io/projects/spring-cloud-dataflow

ence in BDA projects. The exclusion criteria were (i) There are inconsistent answers (i.e., self-contradictory), and (ii) respondents that answered less than 50% of the questions.

11.3.2 Survey design

This survey is classified as descriptive research because: 1) This survey was planned and structured, and 2) the information collected can be inferred statistically over a population. This type of research uses closed-ended questions to understand opinions or attitudes by a group of people on a specific topic. This survey is a self-administered questionnaire, where a research participant is given a set of questions to answer via a web-based questionnaire. The survey questions are publicly available[9] in the respondent's native language, i.e., Spanish. We included a self-guided workshop with an opening paragraph to introduces the purpose, case study, and scenarios needed to answer the instrument.

11.3.3 Developing of survey instrument

Our questionnaire consisted of nine sections and 24 questions written in Spanish, the participant's native language. The first section of the survey is composed of demographic questions asked for job, role, level of education, and subjects' experience. These questions also asked for company information like industry sector, size, experience, and maturity. This first section helped us to understand the participant's background. The remaining eight sections correspond to the eight scenarios used to collect data about the general perception of ACCORDANT against SCDF.

11.3.4 Evaluating the survey instrument

The questionnaire was reviewed externally by two other researchers, and they checked the content, meaning, and understandability. Additionally, four practitioners on BDA projects answered a pilot to refine the instrument and estimate the time needed to complete the survey.

11.3.5 Data analysis

Data analysis was done through the following steps: (i) collection of responses into a single spreadsheet, (ii) analysis of the spreadsheet using descriptive statistics for quantitative answers

[9] https://forms.gle/9VfkrCFWzPsKXGkPA

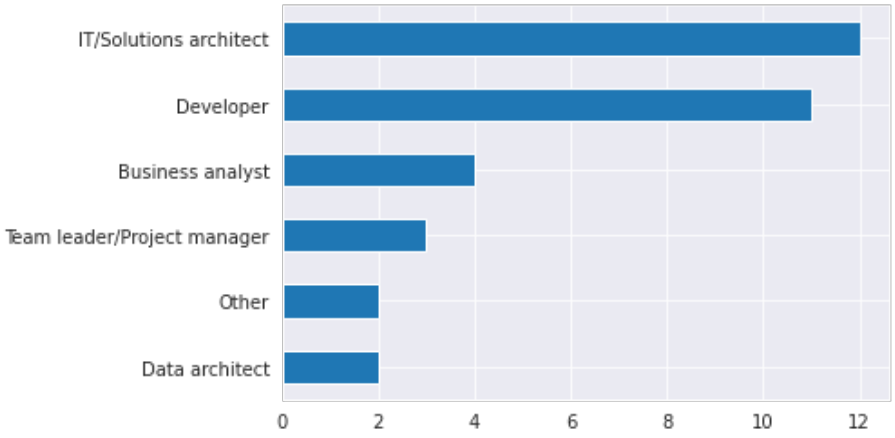

Figure 11.3.1: Respondent's job

for each given response, and (iii) identification of key findings from results of the statistical analyses. The raw answers are publicly available[10].

11.3.6 Survey results

This section reports the survey results based on collected data. Thirty-four answers were collected. The first section summarizes the background data, and then we analyze ACCORDANT and SCDF performance in eight scenarios.

11.3.6.1 Background data

This subsection describes the background information of the respondents. This background can influence the perception of respondents regarding the architecture scenarios evaluated. This information includes the respondent's role, industry, and experience in BDA projects.

Fig. 11.3.1 describes the respondent's job frequency. The 35.3% (12 out of 34) of respondents are IT or solutions architect, 32.3% are developers, 11.7% business analysts, 8.8% are team leaders or project managers, and 5.8% data architects. The company's sector of the respondents is detailed in Fig. 11.3.2 to understand the business environment in which BDA projects are developed. Technology is the most common industry (44.1%), followed by finance (20.6%) and government (11.7%). Education (8.8%) and commerce (5.8%) continue the list. The remainder sectors that close the list with (2.9%) are transportation, tourism, and telecom. Finally,

[10]https://storage.googleapis.com/ccastellanos/Anonymous-BDAToolsSurvey.csv.zip

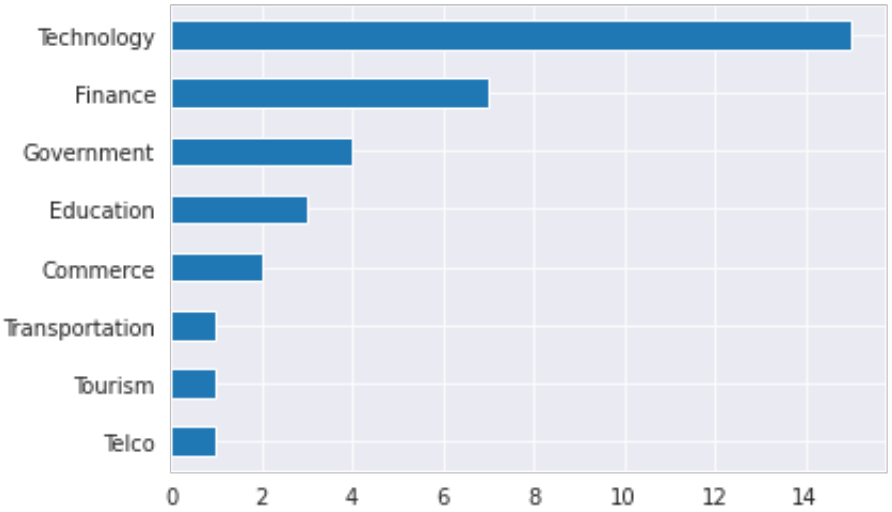

Figure 11.3.2: Company industry

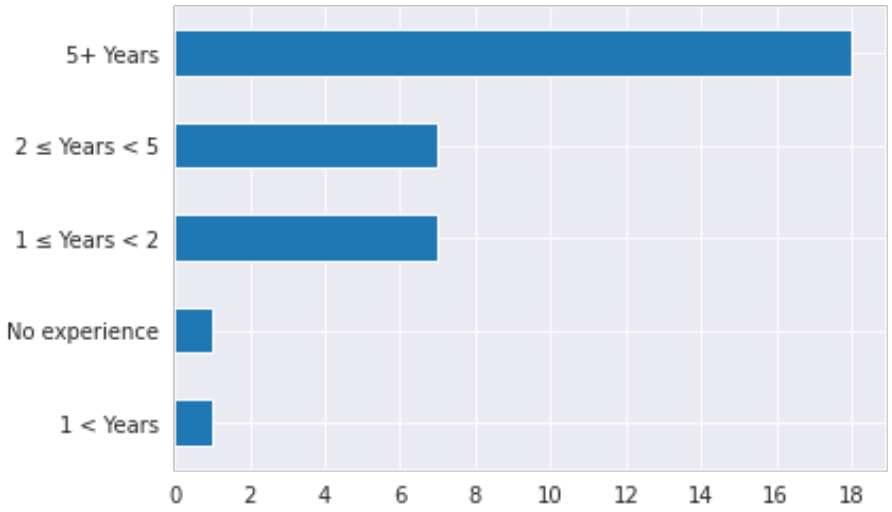

Figure 11.3.3: Respondent's years experience in BDA

Fig.11.3.3 reports the years of experience in BDA projects. Most of the respondents (52.9%) have experience of five or more years. The 20.6% of respondents have between two and five years of experience, in the same proportion are the respondents between one and two years of experience. Finally, 2.9% of respondents have less than one year, and the other 2.9% have no experience.

11.3.7 Tools comparison across architecture scenarios

This section compares the average score (from 1 to 5) per question and tool answered by the respondents in each scenario. We calculate the *t-test* on two related samples of ACCORDANT and SCDF scores to measure whether the average score differs significantly across samples (i.e., the probability that the sample data results occurred by chance). If we observe a large *p-value*, for example, greater than 0.05 (5%), then we cannot reject the null hypothesis of identical average scores. If the p-value is smaller than the threshold (e.g., 0,05), then we reject the null hypothesis of equal averages. Small p-values are associated with large t-test values. The t-test is calculated as:

$$t - test = \frac{\bar{X}_D}{s_D/\sqrt{n}}$$

where $\bar{X}_D$ and s_D are the average and standard deviation of the differences between all pairs of respondents scores. The degree of freedom used is $n1$, where n represents the number of pairs. Table 11.3.1 summarizes questions, average scores, *p-value*, and *t-test*. Figure 11.3.4 depicts the same results ordered by from left to right according the *t-test* value, i.e., significant difference.

From these results, we can highlight the scenarios in which ACCORDANT showed the most significant difference compared to SCDF. Our architectural approach address QS and tactics explicitly; therefore, ACCORDANT presented a higher score regarding scenario 1 (questions S1Q06 and S1Q07). Connectors are first-class citizens in our approach, so the respondents assigned a higher score to ACCORDANT in question S2Q09. In contrast, component specifications for each tool do not show significant differences (S2Q8). ACCORDANT also shows a better scoring regarding technology assignment (S3Q13) due to our technology-neutral proposition and the option to select the target technology to generate code. Given that ACCORDANT offers a broader range of features and attributes in the DV, the specification of computing nodes and deployment strategies are better ranked than SCDF, questions S3Q10 and S3Q11. In deployment scenario 4, although ACCORDANT exhibits a lower score than SCDF

Table 11.3.1: Summary of tools' average score, t-test, and p-value per question

Question	Topic	ACCORDANT	SCDF	T-test	p-value
S1Q06	QS specification	4,65	3,18	6,33	0,00%
S1Q07	Tactics integration	4,50	3,26	4,94	0,00%
S2Q08	Components spec in FV	4,50	4,21	1,66	10,56%
S2Q09	Connectors spec in FV	4,53	3,32	4,91	0,00%
S3Q10	Computing nodes spec in DV	4,65	3,85	3,86	0,05%
S3Q11	Deployment strategies in DV	4,35	3,35	3,57	0,11%
S3Q12	SW Artifacts in DV	4,38	3,88	1,97	5,76%
S3Q13	Technoloy assignment	4,53	3,59	4,29	0,01%
S4Q14	One-time deployment	4,12	4,50	-1,89	6,79%
S4Q15	Multiple deployments	4,26	3,32	3,33	0,22%
S5Q16	Metrics monitoring	4,68	4,71	-0,30	76,80%
S5Q17	QS metrics assessment	4,76	3,12	6,91	0,00%
S6Q18	Decisions in views	4,59	2,82	8,35	0,00%
S6Q19	Decisions in components	4,62	3,32	5,74	0,00%
S7Q20	Scaling-out in DV	4,32	4,50	-1,00	32,46%
S7Q21	Scaling-up in DV	4,41	2,35	6,15	0,00%
S8Q22	New model version spec	4,65	4,00	3,20	0,30%
S8Q23	New model deployment	4,56	4,12	3,27	0,25%

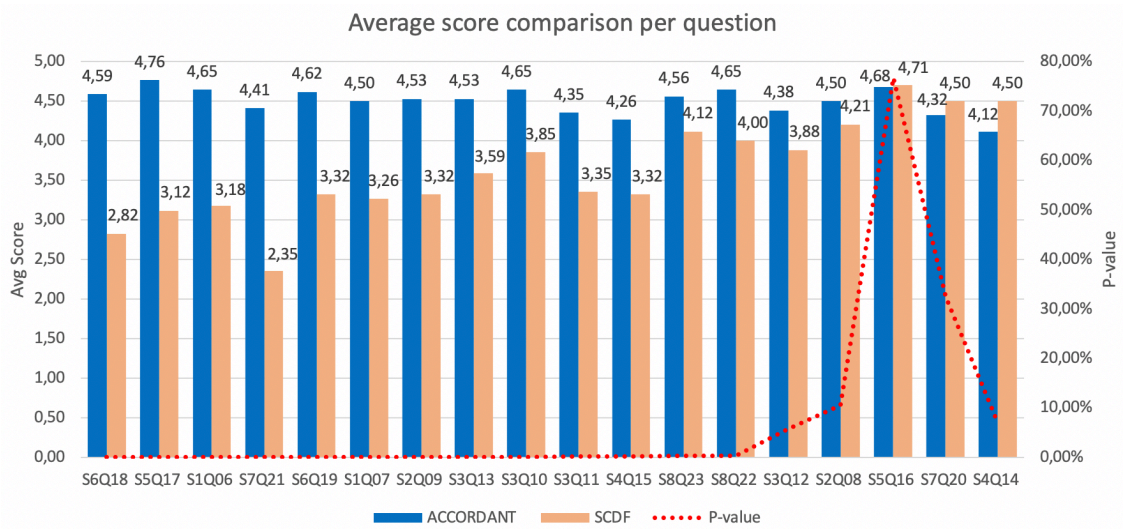

Figure 11.3.4: Comparison of average score and p-value results per tool ordered by t-test

for one-time deployment (S4Q14), ACCORDANT has a better performance with multiple deployments (S4Q15). These results are consistent with the insights found in the previous Section 11.2.

Regarding architectural decisions analysis (S6Q18) and QS metrics assessment (S5Q17), ACCORDANT achieves the highest significant difference and confidence (the lowest p-value) since ACCORDANT formally incorporates QS and decisions from the initial architectural drivers' definition. Besides, ACCORDANT provides a better scaling-up definition because ACCORDANT allows architects to define the computing nodes' types and sizes. ACCORDANT has also shown to improve analytics models upgrading and deployment (S8Q22 and S8Q23, respectively). In contrast, ACCORDANT falls short when compared SCDF in questions such as one-time deployment (S4Q14) and scaling-out (S7Q20), and there are practically no differences in metrics monitoring (S5Q16). The detailed specification in ACCORDANT is required to make one deployment, which is then compensated with multiple further deployments, and this initial effort can explain the results obtained.

We found additional key points reviewing the open-ended questions provided by respondents. Some points to improve ACCORDANT are the currently small number of available technologies, manual tasks required to integrate components, and lack of guidance to enter manual parameters. On the other hand, several comments highlight the importance of accelerating the deployment of BDA applications that traditionally requires cumbersome tasks in daily work.

11.4 Evaluating the ACCORDANT Validator

This section evaluates the architectural mismatch validation presented in Chapter 8 using four case studies. The four case studies address component-connector interaction, infrastructure, and application domain, which make use of the seven constraints presented in Chapter 8, Section 8.3 and detailed in Table 11.4.1.

Table 11.4.1: Case studies and applied constraints

Code	Description	Field	Constraints
CS1	Bus Delay Prediction	Transportation	F14, F12
CS2	NMAC Detection	Avionics	D3, ID5
CS3	Tax Fraud Detection	Finance	F14, FD5
CS4	NMAC Risk Analysis	Avionics	IF1, FD3

We choose these constraints because they were actually faced in practice during the design

and implementation of the following real-life case studies. This section will detail four case studies and extracts of their specifications using ACCORDANT, but the complete specifications can be found at ACCORDANT use cases repository[11].

Listing 11.3: Extract of CS1'S FV

```
1  FunctionalView CS1FVModel use project CS1{
2    Components {
3      Ingestor DataRequest {
4        procModel:STREAM delivery:EXACTLY_ONCE ...
5        ports: {Port busdata:PROVIDED
6          fields:{
7            stopId:Integer order 1,
8            routeId:Integer order 2,
9            weekDay:Short order 3,
10           timeWindow:Short order 4,
11           avgDelay:Float order 5
12   }}},
13     Transformer StreamAvg {
14       procModel:STREAM delivery:AT_LEAST_ONCE ...
15       ports:{..., Port sAvg_req:REQUIRED
16         fields:{
17           stopId:Integer order 1,
18           routeId:Integer order 2,
19           weekDay:Short order 3,
20           timeWindow:Date order 4,
21           avgDelay:Float order 5
22   }}}, ...
23   Connectors {
24     Event InMSG { delivery: EXACTLY_ONCE ...
25       roles : {
26                 Role in_ing:IN  -> busdata,
27                 Role sout_ing:OUT -> sAvg_req,
28                 Role bout_ing:OUT -> bAvg_req
29     }}, ...
30   }}}
```

11.4.1 CS1 - Bus delay prediction

Case study 1 is applied in transportation analytics based on the work presented by Perez et al. in [76]. A lambda architecture that combines batch and real-time processing is implemented using SaaS platforms on three different public cloud providers in this case study. An ingestor collects data from bus trips every 60 seconds to be aggregated by a transformer in batch and speed layers to calculate average delays per window time. Finally, a sink component stores the outputs in the serving layer. An extract of the FV specification in ACCORDANT DSL of this

[11] https://github.com/kmilo-castellanos/accordant-usecases

case study is detailed in Listing 11.3. An ingestor (*DataRequest*) and transformer (*StreamAvg*) components are defined from lines 3 to 22. The ports *busdata*, *SAvg_req* of these components are specified as *PROVIDED* and *REQUIRED* respectively, lines 5 and 15, along with their fields detailing name, data type and order. These ports are connected through the event connector *InMSG* (lines 26-27).

In this version, the *timeWindow* data types of provided and required ports differ (line 9: *Short*, and line 18: *Date* respectively), so these ports present a mismatch detected by intra-view constraint *F14* (see Listing 8.1). In addition, a delivery guarantee mismatch between components *DataRequest* and *StreamAvg* is also detected by constraint *F12* because they are directly connected, and *DataRequest* specifies exactly-once guarantee (line 4), but *StreamAvg* only offers at-least-once guarantee (line 14).

11.4.2 CS2 - Near mid-air collision (NMAC) detection

Case study 2 addresses an avionics safety use case to detect near mid-air collision (NMAC) in near real-time, based on a previous implementation presented in [17]. Listing 11.4 shows the architectural project and drivers detailing QS *CS2_QS1* (lines 3-6) and *CS2_QS2* (lines 7-10). These QS define a quality attribute (Performance), measure (throughput), expected values (1,200 and 1,600), and metric (messages per second, MSG/S). To achieve these QS, the analyzed quality scenarios are defined (lines 11-18) along with their architectural decisions: *CS2_D1* and *CS2_D2* (lines 14 and 17 respectively). Architectural decisions will be referenced during architectural design.

Listing 11.4: Extract of CS2's architectural drivers

```
1  Project CS2{
2    QScenarios {
3      QS CS2_QS1 {
4        QA:PERFORMANCE ...
5        measure: THROUGHPUT 1200 in MSGS
6      },
7      QS CS2_QS2 {
8        QA:PERFORMANCE ...
9        measure: THROUGHPUT 1600 in MSGS
10     }}
11   analyzedQScenarios{
12     AQS CS2_AQS1 of CS2_QS1 {
13       ArchDecisions{
14         Decision CS2_D1{...}}},
15     AQS CS2_AQS2 of CS2_QS2 {
16       ArchDecisions{
17         Decision CS2_D2{...}}}
```

```
18  ...}}}
```

The FV model, detailed in Listing 11.5, comprises an *ADSBReq* (lines 3-6) component which consumes flights data from an ADS-B service (Automatic dependent surveillance– broadcast). Flight's data are pushed in a message queue to be consumed by the *NMACDetector* (lines 7-12) component, which classifies NMAC alerts using a regression tree model (line 8). The NMAC alerts are finally persisted into a MongoDB database by a sink component *NMACWriter* (lines 13-16). These components are connected via *Ports* and *Roles* attached to *Event* connectors, lines 17-21. To align FV and architectural drivers, architectural decisions are associated to components *NMACDetector* (line 9) and *NMACWriter* (line 15).

Listing 11.5: Extract of CS2's FV

```
1  FunctionalView CS2FVModel use project CS2{
2    Components {
3      Ingestor ADSBReq {
4        procModel: MICROBATCH type:HTTP ...
5        ports: {Port adsb:PROVIDED ...}
6      }},
7      Estimator NMACDetector {
8        procModel: MICROBATCH pmml:"...DTree.PMML"
9        decision: CS2_D1
10       ports:{Port nmacIn:REQUIRED,
11         Port nmacOut:PROVIDED ...
12     }},
13     Sink NMACWriter {
14       procModel: MICROBATCH conn:"mongodb://..."
15       decision: CS2_D2
16   }}}, ...
17   Connectors {
18     Event InMSG { delivery:EXACTLY_ONCE ...
19       roles : {Role in:IN -> adsb,
20                 Role out:OUT -> nmacIn,
21     }}, ...
22   }}}
```

To deploy the CS2's application, a DV model is designed to describe the underlying technology infrastructure as detailed in Listing 11.6. This DV model includes a set of artifacts *NMACArtifact* and *NMACWriterArt* which are associated to components *NMACDetector* and *NMACWriter* from FV to be deployed, lines 4-10. In addition, devices (virtual or physical computing nodes) and their specific computing resources are defined in line 12. The deployment of artifacts *NMACArtifact* and *NMACWriterArt* are specified detailing the number of replicas (lines 14 and 24), and the pods managed. *NMACArtifact* uses a Spark cluster, hence Spark workers pod is configured in lines 16-22, associating the device *a* and the execution environment *SparkWEnv* in lines 17-19. *SparkWEnv* environment specifies a containerized image to

be installed (line 20), requirements for CPU (30%) and memory (3 GB) in line 21, and the deployed artifact *NMACArtifact* in line 22.

This DV specification illustrates architectural mismatches *D3* and *ID5*. The infrastructure constraint *D3* is violated because memory requirements for all Spark workers add up to more than 8 GB available in the device *a*. Therefore it is necessary to reduce the memory requirements or to increase the available resources. This mismatch is identified by intra-view constraint *D3* (see Listing 8.1), and this constraint has been shown to be useful in practice since when this deployment is executed in Kubernetes, the Kube-controller makes several attempts to deploy such pods, failing due to the lack of resources to provision. As a result, this constraint helped us detect infrastructure resources mismatch earlier at design time, avoiding delaying the inconsistency detection until the deployment time. Besides, an inter-view mismatch is also detected by the constraint *ID5* between components *NMACArtifact* and *NMACWriterArt*, regarding their QS (1,200 and 1,600 MSG/S respectively, see Listing 11.4) and their replicas. The total throughput required by three *NMACArtifact*'s replicas (3,600 MSG/S) cannot be satisfied by two *NMACWriterArt*'s replicas (3,200 MSG/S).

Listing 11.6: Extract of CS2's DV

```
1  DeploymentView CS2DVModel
2  use project CS2
3  use functionalView CS2FVModel {
4    artifacts{ ...
5          Artifact NMACArtifact {
6              component : NMACDetector
7              technology: Spark},
8          Artifact NMACWriterArt {
9              component : NMACWriter ...}
10   }
11   devs{
12     Device a {host: "a" cpu: 4 memory: 8 ...}}
13   deployments{
14     Deployment SparkWorkerDep {replicas: 3
15       pods {
16         Pod SparkWPod{
17           device: a
18           envs{
19             ExecEnv SparkWEnv{
20               image: "ramhiser/spark:2.0.1"
21               cpu_req: 0.3 memory_req: 3.0
22               deployedArtifacts{NMACArtifact}
23     ...}}}}}
24     Deployment MongoDriverDep {replicas: 2
25       pods {
26         Pod MongoDriver{
27           device: c
```

```
28            envs{
29              ExecEnv JVM{ ...
30                deployedArtifacts{NMACWriterArt}
31      ...}}}}}}
32    }
```

11.4.3 CS3 - Tax fraud detection

Case study 3 is based on the tax fraud detection application developed in the CAOBA alliance[12], which is currently deployed in the Department of Finance of Bogotá, Colombia. This application loads data from urban delineation tax returns, building permits, and cadastral to detect under-reporting tax declarations using spectral clustering. More information about how this clustering model was built is detailed in [32]. The FV specification is detailed in List 11.7. Three ingestor components *TaxReturns*, *BuildPermits*, and *Cadastral* (line 3) collect the data to be transformed within the ETL pipeline. This pipeline is comprised by the following transformers which make joining, filtering and pre-processing of collected data: *LicenseJoin*, *HistClean*, *ViewsPrecomp*, and *CuratorDeliJoin* (lines 5–13). Finally, the spectral clustering component *SpecClustering uses these pre-processed data* (lines 14–22) to create clusters of works based on their economic and physical features. The output of this clustering assignment is stored in a MongoDB collection by the sink component *ModelResult*.

Listing 11.7: Extract of CS3's FV

```
1  FunctionalView CS3FVModel use project CS3{
2    Components {
3      Ingestor TaxReturns, BuildPermits, Cadastral ...,
4      Sink ModelResult ...,
5      Transformer CuratorDeliJoin {
6        procModel:BATCH ...
7        ports:{..., Port CurDelProv:PROVIDED
8          fields:{
9            zone: Short order 1,
10           stratum: Short order 2,
11           floors: Short order 3,
12           area: Float order 4,
13           built_area: Integer order 5 },
14     Estimator SpecClustering {
15       procModel:BATCH algorithm: SpectralClustering
16       ports:{Port CurDelReq:REQUIRED
17         fields:{
18           zone: Short order 1,
19           stratum: Short order 2,
20           floors: Short order 3,
```

[12]http://alianzacaoba.co/

```
21            area: Float order 4,
22            built_area: Float order 5 }
23    }}}, ...
24    Connectors {
25      ProcCall InvokeClustering {
26        roles : {
27          Role InClust:IN  -> CurDelProv,
28          Role OutClust:OUT -> CurDelReq,
29      }}, ...
30    }}}
```

An extract of the DV definition is detailed in 11.8. The artifact *SpecClusteringArtif* (lines 5–7) specifies that component *SpecClustering* is implemented using Spark, and deployed within pod *SparkWPod*, execution environment *SparkWEnv*, see line 18.

This version of the CS3's specification exhibits two architectural mismatch: incompatible port structures (constraint *F14*), and unsupported algorithms by selected technology (constraint *FD3*). *F14* is violated because the fields of connected ports *CurDelProv* and *CurDelReq* differs in terms of datatypes, see Listing 8.1. The last field *built_area* is *Integer* in *CurDelProv* (line 13), but *Float* in *CurDelReq* (line 22). In addition, constraint *FD3* is also infringed, since none result is returned by the SPARQL query detailed in Listing 8.3, when executed over ACC-OWL Ontology for *?param1= 'Spark'* and *?param2='SpectralClustering'*. Indeed, the algorithm *Spectral Clustering* is not directly supported by Spark-MLib technology at the moment of this writing. As a result, this application was implemented using Sci-kit Learn Python library which does support *Spectral Clustering* algorithm.

Listing 11.8: Extract of CS3's DV

```
1   DeploymentView CS3DV
2   use project CS3
3   use functionalView CS3FVModel {
4     artifacts{
5       Artifact SpecClusteringArtif {
6         component : SpecClustering
7         technology: Spark } ...
8     }
9     devs{...}
10    deployments{
11      Deployment SparkWorkerDep {
12        replicas: 3
13        pods {
14          Pod SparkWPod{
15          envs{
16            ExecEnv SparkWEnv{
17              deployedArtifacts{
18                SpecClusteringArtif, ...}
19                image: "ramhiser/spark:2.0.1"
```

```
20                cpu_req: 0.3
21                ports [8081]
22                commands ["/spark-worker"]
23  ...}
```

11.4.4 CS4 - NMAC RISK ANALYSIS

Case study 4 is focused on avionics safety risk analysis and extends the use case presented in [19]. This CS4 identifies clusters of risk levels based on historic ADS-B data in a batch processing mode. The CS4's architecture drivers are detailed in Listing 11.9. The processing model constraint *CS4_CN1* (lines 5-7) restricts to batch processing. Listing 11.10 details the FV. ADS-B data are stored as JSON files in a distributed file system. The ingestor component *JsonReader* (lines 3-5) loads ADS-B data and calls *NMACDetector* component (lines 6-8), which computes alert levels. Then, the NMAC alerts are sent to a clustering estimator *NMACCluster* (lines 9-11) to be assigned to a specific cluster. Finally, clustering estimation results are stored back in the file system by the *NMACCluster* (lines 12-14).

Listing 11.9: Extract of CS4's architectural drivers

```
1  Project CS4{
2    QScenarios {...}
3    analyzedQScenarios {...}
4    constraints {
5      Constraint CS4_CN1{
6        type: PROC_MODEL = "BATCH"
7      }}
8  ...}
```

Listing 11.10: Extract of CS4's FV

```
1  FunctionalView CS4FVModel use project CS4{
2    Components {
3      Ingestor JsonReader {
4        procModel:BATCH type:HDFS ...
5      },
6      Estimator NMACDetector {
7        procModel:MICROBATCH pmml:"...DTree.PMML"
8      },
9      Estimator NMACCluster {
10       procModel:BATCH pmml:"...Cluster.PMML"
11     },
12     Sink HDFSWriter {
13       procModel:BATCH type:HDFS
14       conn: "hdfs://..."
15   }},
16   Connectors {...}
17 }
```

Listing 11.11 details two DV artifacts of CS4. Artifact *NMACClusterArt* is implemented in *Spark* (lines 5-7), while artifact *HDFSWriterArt* uses *Sqoop* as target technology. Deployment and device specifications are not detailed in this extract because they are not required to illustrate the violation of constraints *IF1* and *FD5*. Constraint *IF1* (see Listing 8.2) validates processing models, and it does not hold for component *NMACDetector*, since its processing model (*MICROBATCH*, line 7 in Listing 11.10) is different from provided on the inputs (*BATCH*, line 6 in Listing 11.9). On the other hand, *FD5* uses a SPARQL query over ACC-OWL to check compatibility between technologies of connected components (see Listing 8.3). In this case, technologies *Spark* and *Sqoop* specified for artifacts *NMACClusterArt* and *HDFSWriterArt* in lines 7 and 10 respectively, have no a *techCompatibleWith* property between them on the ontology. Indeed, checking Sqoop specifications, there are not Spark a direct connector for Spark at the time of this writing, which could implies additional glue code to integrate these technologies.

Listing 11.11: Extract of CS4's DV

```
1  DeploymentView CS4DVModel
2  use project CS4
3  use functionalView CS4FVModel {
4    artifacts{ ...
5      Artifact NMACClusterArt {
6        component : NMACCluster
7        technology: Spark},
8      Artifact HDFSWriterArt {
9        component : HDFSWriter
10       technology: Sqoop}
11   }
12   devs{...}
13   deployments{...}
14 }
```

11.4.5 Discussion

The generalization of our work has been addressed by using a wide range of constraints and different case study domains. However, additional constraints could emerge when new case studies introducing different validation requirements. In this case, the OCL constraints schema is incremental so that new constraints can be included, and some other constraints can be disabled. Another critical aspect to consider is the inclusion of different architectural views, such as concurrency, development, operational, and information. As shown in this work, these views offer enough substance to be analyzed and discussed. In the same line of completeness, we acknowledge that presented ontology is not comprehensive in terms of technologies since each day brings us new technologies and features. Therefore, technology consistency validation is bound

to the catalog context, and this catalog is a live artifact that evolves depending on the architecture team's domain and interests.

We could take advantage of mismatch detection and prevention in practice during this approach applications in real-life projects. Mismatch detection in design phases has been addressed in advance, reducing the time invested in redesigning and refactoring in further stages. Specifically, the detection of infrastructure resources mismatch and technology incompatibilities has been frequently used since traditional approaches like Kubernetes involve error-prone manual checking. Another research line is related to enable ACCORDANT ADL integration with broader used languages such as UML, ACME, DARWIN, or π-ADL through extensions or specializations.

11.5 Results

In this chapter, we have evaluated our approach presented in chapters 6, 7, 8, and 9 using a survey and case studies. This evaluation tests the three hypotheses stated in Section 1.4:

H1: Including a DSM approach reduces the deployment time of BDA solutions. According to the results obtained in three case studies and a survey with 34 respondents, we have shown that using our DSM approach, the BDA development and deployment time can be reduced when multiple iterations are involved. Despite the initial specification effort, subsequent deployments are accelerated by reusing high-level abstractions and ode generation. In addition, we found that the metrics assessment of BDA applications is also accelerated by including instrumentation directly aligned to quality scenarios (QS). We have also found that ACCORDANT facilitates the deployment of analytics model upgrades compared to a similar approach.

H2: Including a DSM approach improves the architectural design and analysis of BDA applications. In the survey and case studies, we have found that our DSM approach allows architects to design integrated models using multiple viewpoints, thus enhancing QS, connectors, and deployment specifications. Also, the definition and analysis of architectural decisions exhibited a significant improvement compared to traditional approaches.

H3: Implementing a DSM approach and constraints enables architects to check architectural mismatches in BDA applications. We have presented an architectural mismatch detection based on OCL constraints and semantic queries to enable architects to validate BDA architecture models. This validation improves the quality of BDA architectures by checking domain-specific constraints and technology compatibility.

12
Conclusions

In this chapter, we first synthesize and conclude all the contributions of this thesis, revisit the challenges and how we addressed each of them. Then, we discuss some perspectives for future research.

This research has identified challenges in BDA development and deployment related to heterogeneous tooling, communication gaps between stakeholders from different domains, competing quality attributes in BDA environments, and lack of deployment procedures. Besides, the current related work falls short in considering software architecture practices, QS-driven design, interoperability, separation of concerns, and architecture validation. The surveys have reported challenges reported by practitioners during real-life project development, such as heterogeneous tools and complex technology integration. We also have identified that the analytics development process in the data lab falls short of software development and architecture practices since this development is often considered as prototyping.

We have presented a DSM approach in the BDA context to reduce development and deployment time and improve architectural design, analysis, and validation. Our approach is composed of a reference architecture in BDA (ACCRA), a method, and a DSM framework. We have validated our work using a survey and case studies. The results suggest improvement in

time, design, analysis, and QS monitoring compared to previous approaches. ACCORDANT has been shown to improve the deployment gap, especially for projects with many deployments when multiple iterations award the initial specification effort. The ACCORDANT capacity to reuse architecture decisions and functional specifications in multiple deployment views enables architects to seamlessly test and evolve BDA architectures. ACCORDANT favors architecture design, analysis, and monitoring by including drivers and associating them to architecture artifacts. In addition, ACCORDANT provides a quality improvement for BDA architectures via code generation and properties validation.

Our evaluation process could confirm the three defined hypotheses *H1, H2* and *H3* using the EVAL2 and EVAL3 of the DSR methodology. *H1*: The experimentation showed that ACCORDANT reduces the deployment time of BDA solutions when multiple iterations are involved. *H2*: The results of the experimentation and survey suggest that ACCORDANT improves the architectural design and analysis of BDA applications. Finally, *H3*: The case studies that implemented constraint validation showed that the ACCORDANT Validator enables architects to check architectural mismatches in BDA applications.

Some challenges remain, such as the learning curve of ACCORDANT syntax and semantic, even though the visual DSL developed. The ACCORDANT tools also have areas to improve due to their scope as a prototype. Some tasks must still be done manually to deploy the generated code, integrate connectors and components, and connect the monitoring tool. Practitioners who have interacted with ACCORDANT web tooling reported a lack of guidance to specify parameters and keywords during the modeling tasks. The ACCORDANT technology-specific templates support code generators that promote flexibility, but the current availability of technologies in ACCORDANT is short. So to make ACCORDANT more widely used, additional technology templates should be implemented. The initial specification cost makes ACCORDANT unsuitable for projects with one-time deployment or for simple applications that do not address QS and architectural drivers.

12.1 Summary of final contributions

- A survey which identifies practices, quality attributes, tooling, and challenges in BDA development and deployment. The results of this work are published in [17].
- A mapping study that synthesizes the state-of-the-art and gaps in BDA development and deployment.

- ACCRA, a reference architecture that included patterns and tactics catalog to guide the architecture design of BDA applications. The results of this work are published in [20].
- A method to design and deploy BDA architectures integrating perspectives of business experts, architects, and data scientists. The results of this work are published in [19]
- ACCORDANT, a domain-specific modeling approach composed by viewpoints to design and deploy BDA solutions. The results of this work are published in [16, 18, 21].
- An architecture description language (ADL) conforms to ACCORDANT metamodel to specify architectures via textual and visual tooling. The results of this work are published in [16, 18, 21].
- A set of architectural mismatch definitions expressed as inter-view and intra-view constraints on ACCORDANT to validate constraints in BDA architectures. The results of this work are submitted to the Journal of Systems and Software.
- A complete implementation of constraint specifications expressed in OCL and a semantic query language (SPARQL) to validate ACCORDANT models. The results of this work are submitted to the Journal of Systems and Software.
- A set of case studies applied in our approach to validate our hypothesis. The results of this work are published in [20, 21].
- A survey applied to 34 BDA practitioners to evaluate the ACCORDANT MF in eight architecture scenarios.

12.2 Research perspectives

This work opens new research lines and future work in multiple areas, and the following sections explore the potential extensions and future directions.

12.2.1 Support for additional architecture views

We have focused the architecture specification on two viewpoints: functional and deployment. Future research could evaluate additional viewpoints relevant to BDA applications, such as information and concurrency viewpoints. The specification of information assets integrates ingestors and sinks with data sources. Information viewpoint also enables ACCORDANT to reason about

data volume, velocity, and structure to make better decisions and holistic designs. The concurrency view would describe processes and execution to identify the parts of the system that can execute concurrently. This view plays a key role when distributed, and parallel processing is defined and analyzed to identify deadlocks or livelocks.

12.2.2 Integration with well-known ADLs

Another research line is related to enable ACCORDANT integration with broader used ADLs such as ACME, DARWIN, or π-ADL through extensions or specializations. This integration would facilitate the use of ACCORDANT and improve the validation of general architecture constraints that previous ADLs support. Another benefit in this line is validating time restrictions using temporal logic implemented in previous approaches.

12.2.3 Recommender systems for architecture design

ACCORDANT stores architecture decisions, patterns, tactics, technologies, and deployments formally as model versions. Future work can collect the application metrics associated with model versions to recommend architecture mechanisms that have previously shown better performance. Such recommendations can match architecture characteristics, machine learning algorithms, technology restrictions, and additional architecture elements.

12.2.4 Technology evolution and architecture erosion validation

The technology consistency validation is bound to the catalog context, and this catalog is a live artifact that evolves depending on the architecture team's domain, interests, and technology evolution. New research lines can explore the automatic updating of technologies catalog based on the vendor documentation. Detecting technology evolution is crucial to identify potential architecture erosion when the technology used in current applications becomes obsolete.

12.2.5 Validation of machine learning models

Our validation was focused on architectural properties, but new studies could address functional validation of analytics models to check the correctness of such models. PMML supports model verification[1] which is based on a set of verification records added to the model to check that results are consistent with an environment where the model was developed. Besides, another approach is to verify the model correctness formally using proof methods.

[1] http://dmg.org/pmml/v4-3/ModelVerification.html

12.2.6 Support additional portable machine learning formats

ACCORDANT supports PMML models to incorporate ML functional specifications. However, other formats could be included to have wider compatibility, such as PFA (Portable Format for Analytics) and ONNX (Open Neural Network Exchange Format). On the one hand, PFA[2] promotes to be more flexible than PMML because it has control structures, a true type system for both model parameters and data, and statistical functions. On the other hand, ONNX[3] is an open format built to represent machine learning models and deep learning models.

12.2.7 Support additional deployments models

We have framed ACCORDANT deployments within container frameworks to promote modularity and portability. However, new studies could extend this work to include IoT (i.e., microcontrollers and microprocessors), embedded systems, edge computing, and serverless technologies. These new deployments may require extending or modifying the metamodels to support new features, such as IoT devices or specific-cloud provider properties for serverless services.

12.2.8 Predict performance behavior from historical metrics

Since ACCORDANT collects performance metrics, future extensions could use historical metrics, QS, architecture decisions, tactics applied, and architecture models to predict the expected behavior for a model version. This prediction is a regression problem that estimates performance metrics such as response time, availability, or modifiability based on the architecture models and historical metrics.

[2] http://dmg.org/pfa/docs/motivation/
[3] https://onnx.ai/get-started.html

A

Appendices

A.1 Abbreviations

Table A.1.1: Abbreviations

ADD	Attribute Driven Design Method
ADS-B	Automatic Dependent Surveillance—Broadcast
ATAM	Architecture Trade-off Analysis Method
BDA	Big Data Analytics
DSL	Domain Specific Language
DSM	Domain Specific Model
DV	Deployment View
ENSO	El Niño/Southern Oscillation
ETL	Extract, Transform, and Load
FV	Functional View
IaC	Infrastructure as Code
MDE	Model-driven Engineering
MF	Modeling Framework
ML	Machine Learning
NMAC	Near Mid-air Collision
OCL	Object Constraint Language
PFA	Portable Format for Analytics
PMML	Predictive Model Markup Language
RA	Reference Architecture
QS	Quality Scenario
SCDF	Spring Cloud Data Flow
SPARQL	Standard Protocol and RDF Query Language

References

[1] Mohamed Almorsy and John Grundy. Supporting Scientists in Re-engineering Sequential Programs to Parallel Using Model-driven Engineering. In *Proceedings of the 2015 International Workshop on Software Engineering for High Performance Computing in Science*, SE4HPCS '15, pages 1–8. IEEE Press, 2015.

[2] Mohammad Alrifai, Holger Eichelberger, Cui Qui, Roman Sizonenko, Stefan Burkhard, and Gregory Chrysos. QualiMaster Quality-aware Processing Pipeline Modeling. Technical report, QualiMaster Project, 2014.

[3] Vander Alves, Nan Niu, Carina Alves, and George Valença. Requirements engineering for software product lines: A systematic literature review. *Information and Software Technology*, 52(8):806 – 820, 2010. ISSN 0950-5849. doi: https://doi.org/10.1016/j.infsof.2010.03.014.

[4] Sabby Anandan, Marius Bogoevici, Glenn Renfro, Ilayaperumal Gopinathan, and Patrick Peralta. Spring xd: A modular distributed stream and batch processing system. In *Proceedings of the 9th ACM International Conference on Distributed Event-Based Systems*, DEBS '15, page 217–225, New York, NY, USA, 2015. Association for Computing Machinery. ISBN 9781450332866. doi: 10.1145/2675743.2771879. URL https://doi.org/10.1145/2675743.2771879.

[5] 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, 2012. ISSN 0950-5849. doi: https://doi.org/10.1016/j.infsof.2011.11.009. URL http://www.sciencedirect.com/science/article/pii/S0950584911002333.

[6] Matej Artac, Tadej Borovsak, Elisabetta Di Nitto, Michele Guerriero, Diego Perez-Palacin, and Damian Andrew Tamburri. Infrastructure-as-Code for Data-Intensive Architectures: A Model-Driven Development Approach. In *2018 IEEE International Conference on Software Architecture (ICSA)*, pages 156–165. IEEE, apr 2018. ISBN 978-1-5386-6398-1. doi: 10.1109/ICSA.2018.00025.

[7] Cristóbal Barba-González, José García-Nieto, María Del Mar Roldán-García, Ismael Navas-Delgado, Antonio J Nebro, and José F Aldana-Montes. BIGOWL: Knowledge

centered Big Data analytics. *Expert Systems With Applications*, 115:543–556, 2019. doi: 10.1016/j.eswa.2018.08.026.

[8] Kitchenham Barbara and Charters Stuart. Guideline for performing systematic literature reviews in software engineering. Technical report, Keele University and Durham University, 2007.

[9] Len Bass, Paul Clements, and Rick Kazman. *Software architecture in practice*. Addison-Wesley, 2012. ISBN 978-0-321-81573-6.

[10] Len Bass, Ingo Weber, and Liming Zhu. *DevOps: A Software Architect's Perspective*. Addison-Wesley Professional, 2015. ISBN 978-0-13-404984-7.

[11] L Belli, S Cirani, L Davoli, L Melegari, M Mónton, and M Picone. An open-source cloud architecture for big stream IoT applications. 2015. doi: 10.1007/978-3-319-16546-2_7.

[12] Marco Bernardo, Paolo Ciancarini, and Lorenzo Donatiello. Detecting Architectural Mismatches in Process Algebraic Descriptions of Software Systems. In *IEEE/IFIP Conference on Software Architecture*, pages 77–86, Amsterdam, Netherlands, 2001. IEEE.

[13] Marcello M Bersani, Francesco Marconi, Matteo Rossi, and Madalina Erascu. A Tool for Verification of Big-data Applications. In *Proceedings of the 2Nd International Workshop on Quality-Aware DevOps*, QUDOS 2016, pages 44–45, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4411-1. doi: 10.1145/2945408.2945419.

[14] NIST Big Data Public Working Group. NIST Big Data Interoperability Framework: Volume 1, Definitions. Technical report. URL http://dx.doi.org/10.6028/NIST.SP.1500-1.

[15] Cécile Camillieri, Luca Parisi, Mireille Blay-Fornarino, Frédéric Precioso, Michel Riveill, and Joël Cancela Vaz. Towards a Software Product Line for Machine Learning Workflows: Focus on Supporting Evolution. In *International Conference on Model Driven Engineering Languages and Systems*, pages 65—-70, 2016. URL https://hal.archives-ouvertes.fr/hal-01484050/document.

[16] Camilo Castellanos, Dario Correal, and Juliana-Davila Rodriguez. Executing Architectural Models for Big Data Analytics. In Carlos E. Cuesta, David Garlan, and Jennifer Pérez, editors, *Software Architecture*, pages 364–371, Cham, 2018. Springer International Publishing. ISBN 978-3-030-00761-4.

[17] Camilo Castellanos, Boris Pérez, Carlos A. Varela, María del Pilar Villamil, and Dario Correal. A survey on big data analytics solutions deployment. In Tomas Bures, Laurence Duchien, and Paola Inverardi, editors, *Software Architecture*, pages 195–210, Cham, 2019. Springer International Publishing. ISBN 978-3-030-29983-5.

[18] Camilo Castellanos, Carlos A. Varela, and Dario Correal. Measuring performance quality scenarios in big data analytics applications: A devops and domain-specific model approach. In *Proceedings of the 13th European Conference on Software Architecture - Volume 2*, ECSA '19, page 165–172, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450371421. doi: 10.1145/3344948.3344986. URL https://doi.org/10.1145/3344948.3344986.

[19] Camilo Castellanos, Boris Perez, Dario Correal, and Carlos A. Varela. A model-driven architectural design method for big data analytics applications. In *2020 IEEE International Conference on Software Architecture Companion (ICSA-C)*, pages 89–94, 2020.

[20] Camilo Castellanos, Boris Perez, and Dario Correal. *Smart Transportation: A Reference Architecture for Big Data Analytics*, pages 161–179. Springer International Publishing, Cham, 2021. ISBN 978-3-030-60922-1. doi: 10.1007/978-3-030-60922-1_8. URL https://doi.org/10.1007/978-3-030-60922-1_8.

[21] Camilo Castellanos, Carlos A. Varela, and Dario Correal. ACCORDANT: A domain specific-model and DevOps approach for big data analytics architectures. *Journal of Systems and Software*, 172:110869, 2021. ISSN 0164-1212. doi: https://doi.org/10.1016/j.jss.2020.110869. URL http://www.sciencedirect.com/science/article/pii/S0164121220302594.

[22] Humberto Cervantes and Rick Kazman. *Designing software architectures: a practical approach*. Addison-Wesley Professional, 2016. ISBN 978-013-439078-9.

[23] Joseph O Chan. An architecture for big data analytics. *Communications of the IIMA*, 13 (2):1, 2013.

[24] Pete Chapman, Julian Clinton, Randy Kerber, Thomas Khabaza, Thomas Reinartz, Colin Shearer, and Rüdiger Wirth. Crisp-dm 1.0. *CRISP-DM Consortium*, 76, 2000.

[25] Hong-Mei Chen, Rick Kazman, Serge Haziyev, and Olha Hrytsay. Big Data System Development: An Embedded Case Study with a Global Outsourcing Firm. In *Proceedings of the First International Workshop on BIG Data Software Engineering*, BIGDSE '15, pages 44–50, Piscataway, NJ, USA, 2015. IEEE Press. ISBN 978-1-4673-7025-7. doi: 10.1109/BIGDSE.2015.15.

[26] Hong-Mei Chen, Rick Kazman, and Florian Matthes. Demystifying Big Data Adoption: Beyond IT Fashion and Relative Advantage. In *Twentieth DIGIT Workshop*, pages 1–14, Texas, US, 2015. URL http://aisel.aisnet.org/digit2015.

[27] Hong-Mei Chen, Rick Kazman, and Serge Haziyev. Agile Big Data Analytics for Web-Based Systems: An Architecture-Centric Approach. *IEEE Transactions on Big Data*, 2 (3):234–248, sep 2016. ISSN 2332-7790. doi: 10.1109/TBDATA.2016.2564982.

[28] Hong-Mei Chen, Roland Schütz, Rick Kazman, and Florian Matthes. How Lufthansa Capitalized on Big Data for Business Model Renovation. *MIS Quarterly Executive*, 1615 (14):299–320, 2017.

[29] S. J. Clement, D. W. McKee, and Jie Xu. Service-Oriented Reference Architecture for Smart Cities. In *Symposium on Service-Oriented System Engineering*, pages 81—-85, 2017. URL https://ieeexplore.ieee.org/abstract/document/7943295.

[30] Open Data Group. Analytic Executive Challenged to Create and Deploy Models. Technical report, Open Data Group, 2017.

[31] Dataiku. Building Production-Ready Predictive Analytics. Technical report, Dataiku, 2017. Accessed: 2017-07-11.

[32] Daniel de Roux, Boris Perez, Andrés Moreno, Maria del Pilar Villamil, and César Figueroa. Tax fraud detection for under-reporting declarations using an unsupervised machine learning approach. In *Proceedings of the 24th ACM SIGKDD*, page 215–222, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450355520. doi: 10.1145/3219819.3219878.

[33] DICE. DICE – Horizon 2020 Research and Innovation Action, 2018. URL http://www.dice-h2020.eu/.

[34] Holger Eichelberger, Cui Qin, Roman Sizonenko, and Klaus Schmid. Using IVML to Model the Topology of Big Data Processing Pipelines. In *20th International Systems and Software Product Line Conference*, 2016. doi: 10.1145/2934466.2934476.

[35] Arlene Fink. *The Survey Handbook*, volume 1. Sage, 2003. ISBN 9780803959347.

[36] D. Garlan, R. Allen, and J. Ockerbloom. Architectural mismatch: why reuse is so hard. *IEEE Software*, 12(6):17–26, 1995. ISSN 07407459. doi: 10.1109/52.469757.

[37] David Garlan, Robert Allen, and John Ockerbloom. Architectural mismatch: Why reuse is still so hard. *IEEE Software*, 26:66–69, 2009. ISSN 07407459. doi: 10.1109/MS.2009.86.

[38] Dragan Gaševic, Dragan Djuric, and Vladan Devedžic. *Model driven engineering and ontology development*. Springer Science & Business Media, 2009.

[39] 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-2013)*, pages 71–76, Dec 2013. doi: 10.1109/ICITST.2013.6750165.

[40] Marc Gil, Christophe Joubert, and Ismael Torres. Model-driven Engineering IDE for Quality Assessment of Data-intensive Applications. In *Proceedings of the 8th ACM/SPEC*

on International Conference on Performance Engineering Companion, ICPE '17 Companion, pages 173–174, New York, NY, USA, 2017. ACM. ISBN 978-1-4503-4899-7. doi: 10.1145/3053600.3053633.

[41] Abel Gómez, José Merseguer, Elisabetta Di Nitto, and Damian A Tamburri. Towards a UML Profile for Data Intensive Applications. In *Proceedings of the 2Nd International Workshop on Quality-Aware DevOps*, QUDOS 2016, pages 18–23, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4411-1. doi: 10.1145/2945408.2945412.

[42] Ian Gorton and John Klein. Distribution, Data, Deployment Software Architecture Convergence in Big Data Systems. *IEEE SOFTWARE*, 32(3):78–85, 2015. ISSN 0740-7459. doi: 10.1109/MS.2014.51.

[43] Ian Gorton and Klein John. Distribution, Data, Deployment: Software Architecture Convergence in Big Data Systems. Technical report, Software Engineering Institute, 2014.

[44] M. Gribaudo, M. Iacono, and M. Kiran. A Performance Modeling Framework for Lambda Architecture Based Applications. *Future Generation Computer Systems*, jul 2017. ISSN 0167739X. doi: 10.1016/j.future.2017.07.033.

[45] Alex Guazzelli, Michael Zeller, Lin Wen-Ching, and Graham Williams. PMML: An Open Standard for Sharing Models. *The R Journal*, 1(1):60–65, 2009.

[46] Michele Guerriero, Saeed Tajfar, Damian A Tamburri, and Elisabetta Di Nitto. Towards A Model-Driven Design Tool for Big Data Architectures. In *International Workshop on BIG Data Software Engineering*, 2016. doi: 10.1145/2896825.2896835.

[47] K E Harper, J Zheng, S A Jacobs, A Dagnino, A Jansen, T Goldschmidt, and A Marinakis. Industrial analytics pipelines. In *2015 IEEE First International Conference on Big Data Computing Service and Applications*. Institute of Electrical and Electronics Engineers Inc., 2015. doi: 10.1109/BigDataService.2015.38.

[48] Alan R Hevner, Salvatore T March, Jinsoo Park, and Sudha Ram. Design science in information systems research. *MIS quarterly*, pages 75–105, 2004.

[49] Brandon Hill and Matthew Mahowald. The Emergence of Analytic Deployment. Technical report, Open Data Group, 2017.

[50] Yicheng Huang, Xingtu Lan, Xing Chen, and Wenzhong Guo. Towards Model Based Approach to Hadoop Deployment and Configuration. In *2015 12TH WEB INFORMATION SYSTEM AND APPLICATION CONFERENCE (WISA)*, pages 79–84. S China Normal Univ; SE University; Nanjing Univ; CCF; IEEE Comp Soc; IEEE, Institute of Electrical and Electronics Engineers Inc., 2015. ISBN 978-1-4673-9372-0. doi: 10.1109/WISA.2015.65.

[51] IBM. Foundational methodology for data science. http://www-01.ibm.com/common/ssi/cgi-bin/ssialias?htmlfid=IMW14824USEN, 2015. Accessed: 2017-07-11.

[52] International Data Corporation. Big Data and Business Analytics Revenues Forecast to Reach $150.8 Billion This Year, Led by Banking and Manufacturing Investments, According to IDC. Technical report, IDC, 2017.

[53] P Jovanovic, O Romero, A Simitsis, and A Abelló. Requirement-driven creation and deployment of multidimensional and ETL designs. In *Advances in Conceptual Modeling. ER 2012*, pages 391–395. Springer, Berlin, Heidelberg, 2012. ISBN 978-3-642-33999-8. doi: 10.1007/978-3-642-33999-8_47.

[54] Andrey Kashlev and Shiyong Lu. A System Architecture for Running Big Data Workflows in the Cloud. In Ferrari, E and Kaliappa, R and Hung, P, editor, *2014 IEEE International Conference on Services Computing (SCC 2014)*, pages 51–58, 2014. ISBN 978-1-4799-5066-9. doi: 10.1109/SCC.2014.16.

[55] Raul L Katz. El Observatorio De La EconomÍA Digital De Colombia. Technical report, MinTic, 2017.

[56] R. Kazman, M. Klein, M. Barbacci, T. Longstaff, H. Lipson, and J. Carriere. The architecture tradeoff analysis method. In *Proceedings. Fourth IEEE International Conference on Engineering of Complex Computer Systems (Cat. No.98EX193)*, pages 68–78. IEEE Comput. Soc, 1998. ISBN 0-8186-8597-2. doi: 10.1109/ICECCS.1998.706657.

[57] Steven Kelly and Juha-Pekka Tolvanen. *Domain-specific modeling: enabling full code generation*. John Wiley & Sons, 2008. ISBN 978-0-470-03666-2.

[58] Stuart Kent. Model driven engineering. In Michael Butler, Luigia Petre, and Kaisa Sere, editors, *Integrated Formal Methods*, pages 286–298, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg. ISBN 978-3-540-47884-3.

[59] Barbara Kitchenham and Shari Pfleeger. Personal Opinion Surveys. In *Guide to Advanced Empirical Software Engineering*, pages 63–92. Springer, London, 2008. ISBN 978-1-84800-044-5. doi: https://doi.org/10.1007/978-1-84800-044-5_3.

[60] Barbara Kitchenham, O Pearl Brereton, David Budgen, Mark Turner, John Bailey, and Stephen Linkman. Systematic literature reviews in software engineering – A systematic literature review. 51:7–15, 2008. doi: 10.1016/j.infsof.2008.09.009.

[61] John Klein and Ian Gorton. Design assistant for nosql technology selection. In *Proceedings of the 1st International Workshop on Future of Software Architecture Design Assistants*, FoSADA '15, page 7–12, New York, NY, USA, 2015. ACM. ISBN 9781450334389. doi: 10.1145/2751491.2751494. URL https://doi.org/10.1145/2751491.2751494.

[62] Jay Kreps. Questioning the lambda architecture, 2014. URL https://www.oreilly.com/radar/questioning-the-lambda-architecture/.

[63] David Lechevalier, Ronay Ak, Y Tina Lee, Steven Hudak, and Sebti Foufou. A Neural Network Meta-Model and its Application for Manufacturing. In *2015 IEEE International Conference on Big Data*, 2015.

[64] Francesco Marconi, Marcello M. Bersani, and Matteo Rossi. Formal verification of Storm topologies through D-VerT. In *Proceedings of the Symposium on Applied Computing - SAC '17*, pages 1168–1174, New York, New York, USA, 2017. ACM Press. ISBN 9781450344869. doi: 10.1145/3019612.3019769.

[65] S. Martínez-Fernández, P. S. Medeiros Dos Santos, C. P. Ayala, X. Franch, and G. H. Travassos. Aggregating empirical evidence about the benefits and drawbacks of software reference architectures. In *2015 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 1–10, Oct 2015. doi: 10.1109/ESEM.2015.7321184.

[66] Nathan Marz and James (James O.) Warren. *Big data : principles and best practices of scalable real-time data systems*. 2015. ISBN 9781617290343. URL https://www.manning.com/books/big-data.

[67] Radu Mateescu and Flavio Oquendo. π-AAL: An Architecture Analysis Language for Formally Specifying and Verifying Structural and Behavioural Properties of Software Architectures. *SIGSOFT Softw. Eng. Notes*, 31, 2006. doi: 10.1145/1118537.1118541.

[68] Neno Medvidovic. A Classification and Comparison Framework for Software Architecture Description Languages. 1996.

[69] Natalia Miloslavskaya and Alexander Tolstoy. Big data, fast data and data lake concepts. *Procedia Computer Science*, 88(300-305):63, 2016.

[70] Kief Morris. *Infrastructure as code: managing servers in the cloud*. " O'Reilly Media, Inc.", 2016.

[71] Natalya F Noy, Deborah L McGuinness, et al. Ontology development 101: A guide to creating your first ontology, 2001.

[72] National Institute of Standards and Technology. Nist big data interoperability framework: Reference architecture. https://bigdatawg.nist.gov. Online.

[73] Open Data Group. FastScore™ Engine. URL https://www.opendatagroup.com/fastscore-engine.

[74] Openscoring. Openscoring. URL https://openscoring.io/.

[75] Flavio Oquendo. π -ADL : An Architecture Description Language based on the Higher-Order Typed π -Calculus for Specifying Dynamic and Mobile Software Architectures. *ACM Software Engineering Notes*, 29(3):1–14, 2004.

[76] Pedro F. Perez Arteaga, Camilo Castellanos, Harold Castro, Dario Correal, Luis A Guzman, and Yves Denneulin. Cost Comparison of Lambda Architecture Implementations for Transportation Analytics using Public Cloud Software as a Service. In *13th International Conference on Software Technologies - ICSOFT*, pages 889–896, 2018. ISBN 978-989-758-320-9. doi: 10.5220/0006869308550862.

[77] Diego Perez-Palacin, José Merseguer, José I Requeno, Michele Guerriero, Elisabetta Di Nitto, and Damian A Tamburri. A UML Profile for the Design, Quality Assessment and Deployment of Data-intensive Applications. *Software and Systems Modeling*, 18(6): 3577–3614, 2019.

[78] Kai Petersen, Robert Feldt, Shahid Mujtaba, and Michael Mattsson. Systematic mapping studies in software engineering. In *12th International Conference on Evaluation and Assessment in Software Engineering (EASE) 12*, pages 1–10, 2008.

[79] Jim Pivarski, Collin Bennett, and Robert L Grossman. Deploying Analytics with the Portable Format for Analytics (PFA). In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 579–588. ACM, 2016. doi: 10.1145/2939672.2939731.

[80] Bhashyam Ramesh. Big data architecture. In *Big Data*, pages 29–59. Springer, 2015.

[81] Karl Rexer. 2013 Data Miner Survey. Technical report, Rexer Analytics, 2013.

[82] Karl Rexer, Paul Gearan, and Heather Allen. 2015 Data Science Survey. Technical report, Rexer Analytics, 2016.

[83] Nick Rozanski and Eóin Woods. *Software systems architecture: working with stakeholders using viewpoints and perspectives*. Addison-Wesley, 2012.

[84] Per Runeson and Martin Höst. Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering*, 14(2):131, Dec 2008. ISSN 1573-7616. doi: 10.1007/s10664-008-9102-8.

[85] Rok Rupnik and Jurij Jaklič. The Deployment of Data Mining into Operational Business Processes. In Julio Ponce and Adem Karahoca, editors, *Data mining and knowledge discovery in real life applications*, volume 3. Intech, 2009. doi: 10.5772/6460.

[86] Nitin Sawant and Himanshu Shah. Big data application architecture. In *Big Data Application Architecture Q & A*, pages 9–28. Springer, 2013.

[87] Guido Schmutz, Daniel Liebhart, and Peter Welkenbach. *Service-oriented Architecture: An Integration Blueprint: a Real-world SOA Strategy for the Integration of Heterogeneous Enterprise Systems: Successfully Implement Your Own Enterprise Integration Architecture Using the Trivadis Integration Architecture Blueprint*. Packt Publishing Ltd, 2010.

[88] Mary Shaw and David Garlan. *Formulations and formalisms in software architecture*, pages 307–323. Springer Berlin Heidelberg, Berlin, Heidelberg, 1995. ISBN 978-3-540-49435-5. doi: 10.1007/BFb0015251. URL https://doi.org/10.1007/BFb0015251.

[89] Christian Sonnenberg and Jan vom Brocke. Evaluation patterns for design science research artefacts. In Markus Helfert and Brian Donnellan, editors, *Practical Aspects of Design Science*, pages 71–83, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg. ISBN 978-3-642-33681-2.

[90] Julio Sosa and Claudia Jiménez-Guarín. BiDArch: BigData Architect, Generator of Big Data Solution Architectures. In *12th Colombian Conference on Computing*, pages 17–31. Springer, 2017. doi: 10.1007/978-3-319-66562-7.

[91] Arvind K. Sujeeth, Hyouk Joong Lee, Kevin J. Brown, Hassan Chafi, Michael Wu, Anand R. Atreya, Kunle Olukotun, Tiark Rompf, and Martin Odersky. Optiml: An implicitly parallel domain-specific language for machine learning. In *Proceedings of the 28th International Conference on International Conference on Machine Learning*, ICML'11, page 609–616, Madison, WI, USA, 2011. Omnipress. ISBN 9781450306195.

[92] Richard N Taylor, Nenad Medvidovic, and Dashofy Eric M. Software architecture: Foundations, theory and practice. 2010.

[93] The Object Management Group. Object Constraint Language (OCL), 2014. URL https://www.omg.org/spec/OCL/.

[94] Thomas Erl, Wajid Khattak, and Paul Buhler. *Big Data Fundamentals: Concepts, Drivers & Techniques*. Prentice Hall Press, 2016. ISBN 0134291077 9780134291079. URL http://dl.acm.org/citation.cfm?id=2898954.

[95] L Tuovinen, P Laurinen, I Juutilainen, and J Röning. Data mining applications for diverse industrial application domains with smart archive. In *Proceedings of the IASTED International Conference on Software Engineering*, pages 56–61. ACM, 2008. ISBN 978-0-88986-716-1.

[96] Faheem Ullah and Muhammad Ali Babar. Architectural Tactics for Big Data Cybersecurity Analytic Systems: A Review. *Journal of Systems and Software*, 151:81–118, 2019. doi: https://doi.org/10.1016/j.jss.2019.01.051. URL https://arxiv.org/pdf/1802.03178.pdf.

[97] Wil Van Der Aalst. *Process mining: data science in action*. Springer Berlin Heidelberg, 2016. ISBN 978-3-662-49850-7.

[98] Wil van der Aalst and Ernesto Damiani. Processes Meet Big Data: Connecting Data Science with Process Science. *IEEE Transactions on Services Computing*, 8(6):810–819, nov 2015. ISSN 1939-1374. doi: 10.1109/TSC.2015.2493732.

[99] Jos B Warmer and Anneke G Kleppe. *The object constraint language: getting your models ready for MDA*. Addison-Wesley Professional, 2003.

[100] Dennis Wegener and Stefan Rüping. On Reusing Data Mining in Business Processes - A Pattern-Based Approach. 2011.

[101] Michael Weyrich and Christof Ebert. Reference architectures for the internet of things. *IEEE Software*, 33(1):112–116, jan 2016. ISSN 07407459. doi: 10.1109/MS.2016.20.

[102] Rob Wojcik, Felix Bachmann, Len Bass, Paul Clements, Paulo Merson, Robert Nord, and Bill Wood. Attribute-driven design (add), version 2.0. Technical report, CARNEGIE-MELLON UNIV PITTSBURGH PA SOFTWARE ENGINEERING INST, 2006.

[103] Chen Yang, Peng Liang, and Paris Avgeriou. A survey on software architectural assumptions. *The Journal of Systems and Software*, 113:362–380, 2015. doi: 10.1016/j.jss.2015.12.016.

[104] Johannes Zenkert, Mareike Dornhöfer, Christian Weber, Charly Ngoukam, and Madjid Fathi. Big Data Analytics in Smart Mobility: Modeling and Analysis of the Aarhus Smart City Dataset. In *2018 IEEE Industrial Cyber-Physical Systems (ICPS)*, 2018. doi: 10.1109/ICPHYS.2018.8387685. URL https://doi.org/10.1109/ICPHYS.2018.8387685.

[105] Xiaocong Zhou and Peiyan Li. QoS Specification in Software Architecture for QoS-aware Applications. *Journal of Algorithms & Computational Technology*, 4(3):251, 2007.

[106] Ioannis Zozas, Stamatia Bibi, Dimitrios Katsaros, Panagiotis Bozanis, and Ioannis Stamelos. Feature models for big data applications: Modeling big data applications by applying feature models. In *2017 8th International Conference on Information, Intelligence, Systems and Applications, IISA 2017*, volume 2018-January, pages 1–6. IEEE, mar 2018. ISBN 9781538637319. doi: 10.1109/IISA.2017.8316394.