Contents lists available at ScienceDirect

The Journal of Systems & Software

journal homepage: www.elsevier.com/locate/jss

ACCORDANT: A domain specific-model and DevOps approach for big data analytics architectures[☆]

Camilo Castellanos [a,*], Carlos A. Varela [b], Dario Correal [a]

[a] System Engineering and Computing Department, University of Los Andes, Bogota, Colombia
[b] Department of Computer Science, Rensselaer Polytechnic Institute, Troy, NY, USA

ARTICLE INFO

Article history:
Received 8 April 2019
Received in revised form 26 October 2020
Accepted 16 November 2020
Available online 20 November 2020

Keywords:
Software architecture
Big data analytics deployment
DevOps
Domain-specific model
Quality scenarios
Performance monitoring

ABSTRACT

Big data analytics (BDA) applications use machine learning algorithms to extract valuable insights from large, fast, and heterogeneous data sources. New software engineering challenges for BDA applications include ensuring performance levels of data-driven algorithms even in the presence of large data volume, velocity, and variety (3Vs). BDA software complexity frequently leads to delayed deployments, longer development cycles, and challenging performance assessment. This paper proposes a Domain-Specific Model (DSM), and DevOps practices to design, deploy, and monitor performance metrics in BDA applications. Our proposal includes a design process, and a framework to define architectural inputs, software components, and deployment strategies through integrated high-level abstractions to enable QS monitoring. We evaluate our approach with four use cases from different domains to demonstrate a high level of generalization. Our results show a shorter deployment and monitoring times, and a higher gain factor per iteration compared to similar approaches.

1. Introduction

Big data analytics (BDA) applications use Machine Learning (ML) algorithms to extract valuable insights from large, fast, and heterogeneous data. These BDA applications require complex software design, development, and deployment to deal with big data characteristics: volume, variety, and velocity (*3Vs*), to maintain expected performance levels. Specifically, BDA processing takes advantage of cutting-edge technologies and infrastructures that enable distributed stream computing. But the complexity involved in BDA application development frequently leads to delayed deployments (Chen et al., 2016) and hinders performance monitoring (e.g. throughput or latency) (Ranjan, 2014). Regarding the 3Vs, a BDA solution can be constrained to different performance metrics. For instance, real-time stream analytics applications require low latency and flexible scalability based on data volume fluctuation. On the other hand, heavy workloads, which imply batch processing over big data, demand high scalability and fault tolerance to achieve a particular deadline. One of the key goals of software architecture is the design of the system's structures and their relationships to achieve expected quality properties.

The development of BDA solutions involves three knowledge domains: business, analytics, and technology. In the business domain, business experts have to define business goals and quality scenarios (QS) to drive analytics projects. In the analytics domain, these business goals are translated into specific analytics tasks by data scientists. Finally, in the technology domain, software architects make decisions in terms of tactics, patterns, and deployment considerations keeping in mind quality attributes. Stakeholders from different domains face heterogeneous concerns and different abstraction levels. Due to the lack of techniques, and tools to enable articulation and integration of such domains, BDA solutions development presents a high cost and error-prone transition between development and production environments (Chen et al., 2016; Wegener and Rüping, 2010). Though there is a growing interest of companies in big data adoption, real deployments are still scarce ("Deployment Gap" phenomenon) (Chen et al., 2017).

In the same vein, previous surveys (Rexer, 2013; Rexer et al., 2016; Castellanos et al., 2019) have reported low deployment frequency and delayed deployment procedures caused by analytics model translation, lack of tools' interoperability and stakeholders' communication. These pitfalls could be the result of the

[☆] This research was carried out by the Center of Excellence and Appropriation in Big Data and Data Analytics (CAOBA), 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) within contract No. FP44842-anexo46-2015. We would like to make an special acknowledgment to the Taxation Authority of Bogota at the Bogota's Department of Finance (SHD), to the office of telecommunications of the National Planning Department (DNP), and to the CAOBA-Uniandes team, both faculty and students.

[*] Corresponding author.
E-mail addresses: cc.castellanos87@uniandes.edu.co (C. Castellanos), cvarela@cs.rpi.edu (C.A. Varela), dcorreal@uniandes.edu.co (D. Correal).

https://doi.org/10.1016/j.jss.2020.110869
0164-1212/© 2020 Elsevier Inc. All rights reserved.

traditional approach of BDA development where the data scientist produces the models as source code implemented using machine learning-oriented tools which are focused on analytics perspectives within a controlled environment (data lab). On the other hand, software architects have to translate these models into software products which usually implies rewriting code to obtain productive software components deployed on specific IT infrastructures.

This paper proposes ACCORDANT (An exeCutable arChitecture mOdel foR big Data ANalyTics), a DevOps and Domain-Specific Model (DSM) approach to develop, deploy, and monitor BDA solutions bridging the gap between analytics and IT domains. ACCORDANT allows to design BDA applications using QS, functional, and deployment views. A QS specifies a quality attribute requirement for a software artifact to support design and quality assessment. 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 containerization since it offers consistent modularity to facilitate portability, continuous integration, and delivery.

ACCORDANT is validated using four use cases from different domains by designing functional and deployment models, and assessing performance QS. This validation aims to reduce the time of design, deployment, and QS monitoring of BDA solutions. These use cases range from public transportation and avionics safety to weather forecasting, and they include distributed batch, microbatch, and stream processing. Our results indicate improvements in design and (re)deployment times to achieve the expected performance metrics. In summary, the contributions of this paper are as follows:

- A DSM framework to formalize and accelerate iteratively the development and deployment of BDA solutions by specifying architectural functional, and deployment views aligned to QS.
- Three integrated domain-specific languages (DSLs) to specify architectural inputs, component-connector models, and deployments, thus accelerating BDA deployment cycle.
- A containerization approach to promote automation delivery and performance metrics monitoring for BDA applications aligned to QS.
- The evaluation of this proposal applied to four use cases from diverse domains, and using different deployment strategies and QS.

The rest of this paper is organized as follows. In Section 2, we describe the background on DSM, big data analytics, and DevOps. Section 3 reviews related work. Section 4 presents our methodology and proposal overview. Section 5 presents the use cases for experimentation. Section 6 illustrates the steps followed to validate this proposal. Section 7 presents and discusses the obtained results. Finally, Section 8 summarizes the conclusions and future work.

2. Background

This section describes the core concepts in which this proposal is supported: domain-specific modeling, software architecture, big data analytics, and DevOps.

2.1. Domain-Specific Modeling (DSM) and software architecture

Domain-Specific Modeling enables the software to be modular and resilient to changes through the separation of concerns (SoC) principle by specifying technology-agnostic concepts, relationships, and constraints within the domain. An important advantage of DSM is the close mapping problem and solution domains to provide code generation. Moreover, DSM can speed up and optimize the code generated for the specific platform improving productivity. In order to enable code generation, the domain model requires to be narrow, and it is constrained by a language specification, the *metamodel*. Furthermore, due to the narrow metamodel's scope, the models can be read, checked, validated, and interpreted to generate specific implementations. Regarding representations, DSM can be expressed in graphical, textual, or mixed notation according to the domain context. It is possible to embed multiple views or aspects (for example, analytics, software components, and deployment) using different representations that share elements or mappings.

An architecture description language enables architects to express high-level system structure by describing its coarse-grained components and connections among them. These descriptions are contained in *architectural views* to address different concerns, and these views are built based on collection of patterns, templates, and conventions called *Viewpoints* (Rozanski and Woods, 2005). The architectural design is driven by quality scenarios and primary functional requirements through a systematic design method, such as the Attribute-Driven Design method (ADD, Cervantes and Kazman, 2016). ADD starts identifying inputs: QS, functional requirements, and constraints. In each ADD iteration, a design goal is defined from these inputs, and the selection of architectural structures, tactics, patterns, and their application described across views, aims at achieving such goal. A pattern is a standard, known and reusable solution to a common problem in software architecture. Tactics are design primitives to achieve a response for particular quality attributes. Previous studies have collected both patterns (Erl et al., 2016; Marz and Warren, 2015) and tactics (Gorton and Klein, 2014; Ullah and Babar, 2019) to be applied in the BDA domain.

2.2. Big data analytics

In BDA context, data processing models aim at specific application requirements: *batch* to process large stored datasets all at once with high performance, and *stream* processing for an unbounded data flow in (near) real-time. Due to the complexity of deploying and operating BDA solutions integrating a myriad of technologies, complex analytics models, and distributed infrastructure, some research has been done to tackle such complexity by raising the level of abstraction (Gribaudo et al., 2017; Guerriero et al., 2016; Huang et al., 2015).

Due to the wide range of BDA technologies, portability plays a key role to deploy, operate, and evolve BDA applications, and this is where portable standards appear such as Predictive Model Markup Language (PMML)[1] or Portable Format for Analytics (PFA).[2] PMML is the de facto standard proposed by the Data Mining Group that enables portability of analytics models through neutral-technology XML format. PMML allows specifying a set of machine learning models and data transformations along with their metadata.

2.3. DevOps and IaC

According to Bass et al. (2015), DevOps is a set of practices 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

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

[2] http://dmg.org/pfa/

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. IaC promotes the reduction of cost, time, and risk of IT infrastructure provision by offering languages and tools which allow to 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,[3] an open source for automating deployment, scaling, and management of container clusters which offers to decouple application containers from the infrastructure details.

3. Related work

Several works have proposed frameworks to build and deploy BDA applications. We review and compare some of the most relevant works, that comprise building blocks to construct and deploy BDA pipelines. Indeed, some works have tackled DSM to describe functional and deployment viewpoints involving DevOps practices. We summarize and compare the related work reviewed in Table 1, addressing the identified problem and our vision of using separation of concerns (*SoC*), domain-specific modeling and DevOps to deal with the deployment gap.

Table 1 details in each column some features we identify in the related work as follows. *SoC* is a key design principle for us, since the knowledge domains involved in BDA (business, analytics, and IT) have to be tackled from different perspectives (i.e. viewpoints). In terms of analytics domain, cross-industry (*CI*), and technology-neutral models (*TNM*) promote applicability, and BDA portability respectively. Regarding software architecture concepts, QS specification (*QSS*), functional (*FV*), and deployment (*DV*) views allow us to describe orthogonal concerns such as quality scenarios, components-and-connector, and deployment models. Architectural tactics (*AT*) are design decisions that influence the control of a QS response. A target-technology assignment (*TTA*) complements DSM approaches by supporting a predefined technologies set (*P*) or extensible code generators (*C*). Finally, considering the DevOps practices, deployment specification column (*DS*) defines if only a number of instances (*I*) per component or a whole deployment diagram (*D*) can be described. Additional practices that facilitate the deployment and operation processes are considered: continuous deployment (*CD*), QS monitoring (*QSM*), and self-adaptation (*SA*).

Some works have presented DSM to model analytics functions, however, they do not tackle architecture concepts and deployment considerations because they are only focused on functional definitions. Lechevalier et al. (2015) introduce a DSM framework for predictive analytics of manufacturing data using artificial neural networks to generate analytics models. Sujeeth et al. (2011) OptiML, a DSL for machine learning which describes analytics functions using a statistical model which cover a subset of ML algorithms, this analytics functions are analyzed and optimized before the code generation. CloverDX (0000) is a commercial tool to design data transformations and analytics workflows in a visual way integrating external APIs, and including parallel processing in multiple nodes. CloverDX's functional view includes readers, processors, and writers for a predefined set of technologies, but deployment view is not available and distributed processing must be defined with specific parallel nodes in the functional view, which prevents to use the same functional definition in different deployment strategies. Finally, technology-neutral models, performance scenario specifications, and architectural tactics are not supported.

In contrast, we found another group of studies interested in infrastructure concerns of BDA applications leaving aside their functional components. Gribaudo et al. (2017) propose a modeling framework based on graph-based language to evaluate the system's performance of running applications which follow the lambda architecture pattern. This modeling framework allows users to define stream, batch, storage, and computation nodes along with performance indices to be simulated and evaluated, but neither functional BDA application nor real infrastructure provision are provided as a result. Huang et al. (2015) 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 design, deployment, and evaluation of BDA technology infrastructures. However, it leaves out functional analytics models to get an integrated BDA solution.

QualiMaster (Alrifai et al., 2014) 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. The aim of QualiMaster is to maximize the throughput of a given processing pipeline. Similarly, our proposal generates software for BDA applications, but taking as input the analytics specification of a predictive model, and the performance metrics to be achieved. Unlike Qualimaster, our proposal is technology-neutral and cross-industry which enables a more widespread application.

Fastscore (Open Data Group) is a commercial framework to design and deploy analytics models. Analytics components are conventionally developed using a determined programming language or using 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 factor to increase the concurrency of analytics models.

SpringXD (Anandan et al., 2015) 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 approach, analytics processor can be defined through PMML models, but target technologies are limited to a set of predefined options.

DICE project in Guerriero et al. (2016) and Artac et al. (2018) 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 to elements within the application, which are analogous to QS. Perez-Palacin et al. (2019) presented a profile to enable performance and reliability assessment. DICE supports configuration management, service provisioning, and application deployment, but technology-neutral models and architectural tactics are not considered which could hinder portability and design decision tracing. Due to its focus, DICE requires design at very detailed level, specifying different constructs regarding target technologies, but in our proposal, the technology-specific generators transform functional and deployment artifacts to code.

To summarize, the related work approaches reviewed tackle the BDA applications design, but they are not concern about

[3] https://kubernetes.io/

Table 1
Related work.

Work	SoC	Business (Analytics)		Software architecture					DevOps			
		CI	TNM	QSS	FV	DV	AT	TTA	DS	CD	QSM	SA
Lechevalier et al. (2015)		✓	✓		✓							
Gribaudo et al. (2017), Huang et al. (2015)		✓		✓					D		✓	
CloverDX (0000)		✓			✓				I	✓	✓	
OptiML (Sujeeth et al., 2011)		✓	✓		✓			C		✓		
Qualimaster (Alrifai et al., 2014)	✓			✓	✓	✓	✓			✓	✓	✓
FastScore (Open Data Group)	✓	✓	✓		✓			C	I	✓	✓	
SpringXD (Anandan et al., 2015)	✓	✓	✓		✓			P	I	✓	✓	✓
DICE (Guerriero et al., 2016; Artac et al., 2018; Perez-Palacin et al., 2019)	✓	✓		✓	✓	✓		C	D	✓	✓	✓
ACCORDANT	✓	✓	✓	✓	✓	✓	✓	C	D	✓	✓	

deployment architectural decisions. Specifically, only four proposals follow the SoC principle (Alrifai et al., 2014; Open Data Group; Anandan et al., 2015; Guerriero et al., 2016), and among them, only Qualimaster and DICE (Guerriero et al., 2016) offer a deployment viewpoint. From the architecture perspective, tactics and QS specifications are scarcely ever considered. Based on these findings, we argue that our proposal aims to bridge such gaps.

4. ACCORDANT: A DevOps and domain-specific model approach

This proposal aims at offering a high-level approach to design BDA solutions starting from architectural artifacts, instead of source code. Specifically, we propose ACCORDANT (An exeCutable arChitecture mOdel foR big Data ANalyTics) to deal with functional, infrastructure and QS requirements. Our proposal comprises: a design and deployment process, and a DSM framework to support such process. This paper extends metamodel proposed in Castellanos et al. (2018) by aligning ACCORDANT process to ADD, and including architectural inputs, containerization and serverless deployments in DV. Fig. 1 depicts the ACCORDANT's process, which adapts and integrates an architecture design method (ADD) and analytics methodologies.

The steps performed using ACCORDANT modeling framework are framed in solid lines, while the steps made with external tools are represented by dotted lines. ACCORDANT process is iterative and, it is composed of seven steps: the business user defines (1.1) business goals and (1.2) QS which will guide the next steps. (2) The data scientist develops data transformations, build and evaluates analytics models. The resulting analytics models are exported as PMML files. (3) Architect design the software architecture using 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 interweaved to obtain an integrated model. (5) Code generation of software and infrastructure is performed from integrated models. (6) The code generated in the previous step is executed to provision infrastructure and install the software. (7) QS are monitored in operation to be validated, and design adjustments can be made to achieve QS, if necessary.

4.1. Architectural inputs

According to architecture design methods such as Attribute-Driven Design (ADD) (Wojcik et al., 2006), architecture 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 inputs of the architecture design, therefore we include these initial building blocks in the ACCORDANT metamodel along with other concepts defined in ADD. Fig. 2 details the main input building blocks grouped by the architectural input package (*InputPackage*) which contains the elements required to start the architectural design: Quality Scenario (*QScenario*), Analyzed QS (*AnalyzedQS*), *SentivityPoint* and *Tactic*. A *QScenario* determines a quality attribute requirement (i.e. latency, availability, scalability, etc.) for a specific *Artifact*. 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. A QS is analyzed through a *AnalyzedQS*, and sensitivity points. A *SensitivityPoint* is a property of a decision (a set of elements and their relationships within an architectural view) that is critical for achieving the QS, and that such decision is the application of a tactic to a specific application context. Finally, *Tactic* elements synthesize BDA tactics found in Gorton and Klein (2014) and Ullah and Babar (2019) to be applied in an architecture instance, e.g.: dynamic resource allocation, health monitoring, parallel processing, feature selection, etc.

Once *QScenarios*, *AnalyzedQS*, and *SensitivityPoints* are defined in the *step 1.2* of ACCORDANT process, the software architecture is designed in *step 3* and expressed on the views instantiating tactics in a concrete application. These decisions are associated via *SensitivityPoints*, and they will be evaluated against the initial QS to validated whether the architecture is achieving its goal.

4.2. Functional viewpoint (FV)

FV allows us to design analytics pipelines in terms of ingestion, preparation, analysis, and exporting building blocks. *FV* specifies functional requirements of the analytics solution, and the constructs are described in a technology-neutral way as detailed in the metamodel depicted in Fig. 3. FV is expressed in a component-connector structure. Sensitivity points, from architectural inputs, can be associated to components and connectors to represent where architectural decisions have impact regarding the QS. Component metaclasses are specialized in *Ingestors*, *Transformers*, *Estimators*, and *Sinks*. *Estimators* and *Transformers* are the software component realizations of PMML data model and data transformer respectively, and the PMML file defines their behavior. A *Component* exposes required and provided *Port*.

Connectors metaclasses transfer data or control flow among components through an input or output *Roles*. A set of connector types are defined based on the connector's classification proposed by Taylor et al. (2010): *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.

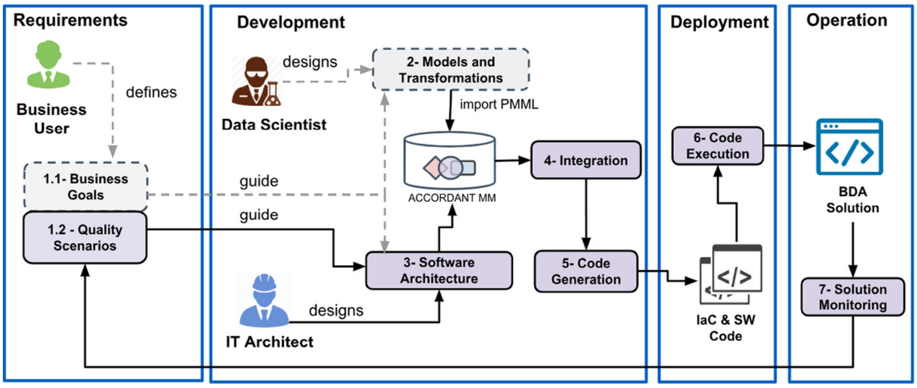


Fig. 1. ACCORDANT process overview.

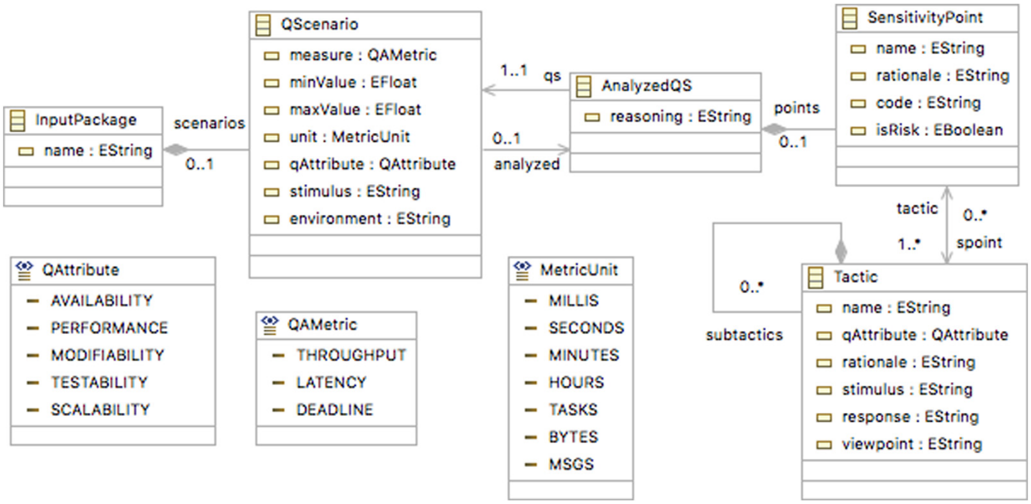


Fig. 2. Excerpt of architecture inputs metamodel.

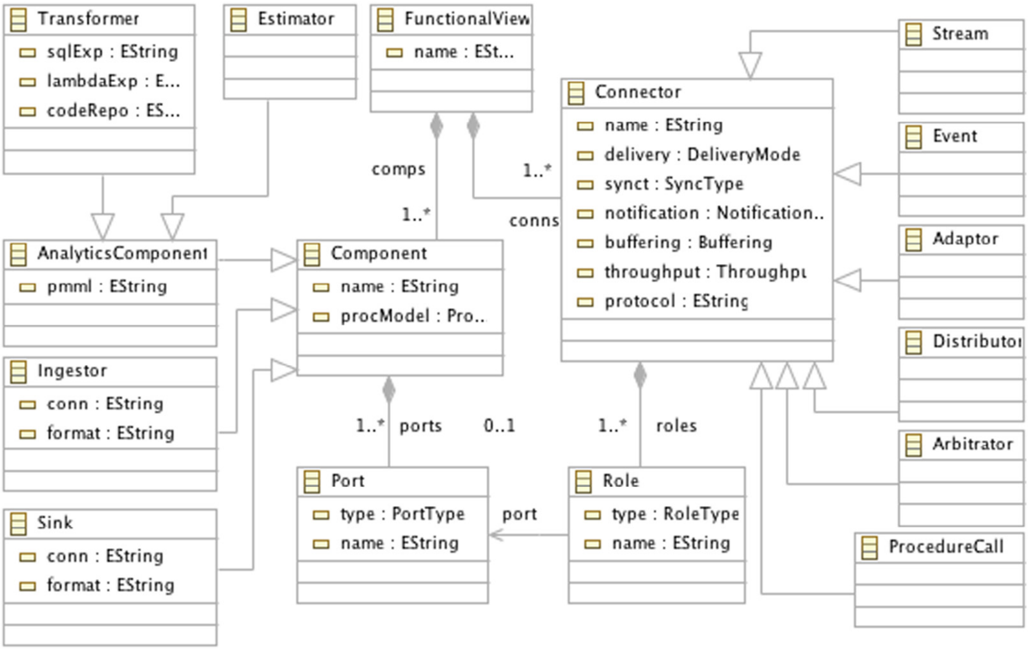


Fig. 3. Excerpt of functional viewpoint of ACCORDANT metamodel.

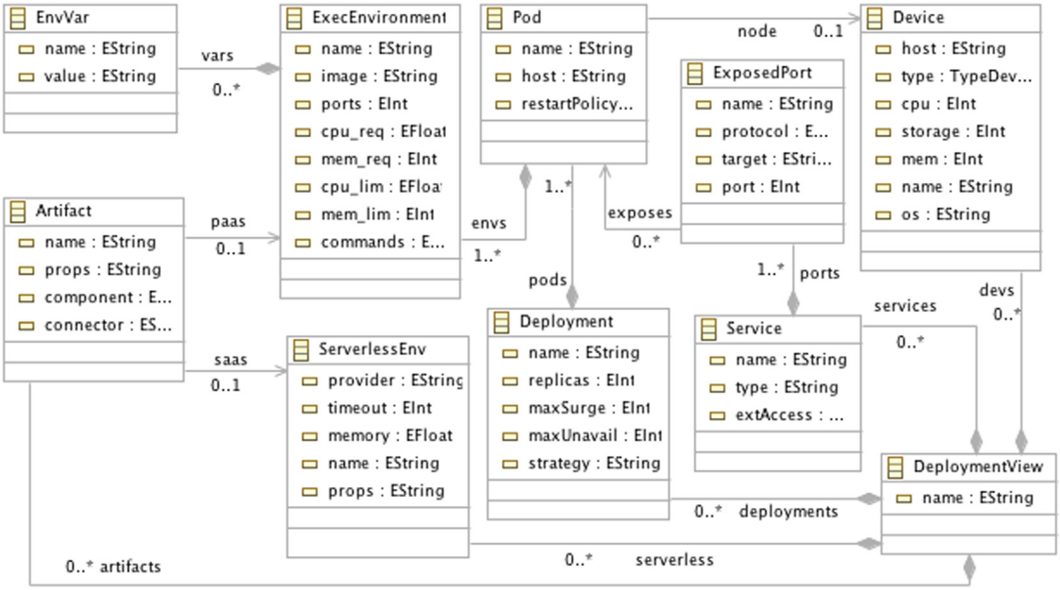


Fig. 4. Excerpt of deployment viewpoint metamodel.

4.3. Deployment viewpoint (DV)

The Deployment viewpoint integrates DevOps practices including containerization, IaC, and serverless computing. The DV specifies how software artifacts (components and connectors) are deployed on a set of computation nodes. The main metaclasses are detailed in Fig. 4. DV metamodel comprises *Pod*, *ExposedPort*, and *Deployment* metaclasses to operationalize BDA applications in a specific technology. 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 QScenarios.

DV contains *Devices*, *Services*, *Deployments*, serverless environments (*ServerlessEnv*), and *Artifacts*. Sensitivity points can be assigned to Deployments and Artifacts to map critical architectural decisions in the DV. 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 resources requirements (CPU, memory). A *Deployment* specifies the desired state for a Pod's group and its deployment strategy, including the number of replicas. *Services* and *ExposedPorts* define the policies, addresses, ports, and protocols by which to access to 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. *Artifacts* correspond to executable or deployable representations of functional elements (i.e. components and connectors from functional view) which can be deployed on either execution or serverless environments.

Once PMML, FV and DV models are designed and integrated, code generation takes place by means of model-to-text transformations. Code generation is twofold: software and infrastructure (IaC) code. On the software side, each component and connector is assigned to a specific technology regarding its constraints specified in the model (processing model, ML algorithm, delivery type, sync type, etc.). Such assignment enables us to generate code for target technology restricted to these constraints. 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 machinery 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. On the IaC side, DV model is transformed into Kubernetes' configuration files (in YAML format) used to create and configure infrastructure over Kubernetes cluster. Kubernetes files contain Nodes, Pods, Deployments, and Services which are executed through Kubectl.

In the last step, the performance metrics of the BDA application are gathered to be compared to initial QS and evaluate the fulfillment of quality requirements. In this step, the architect has to check the outputs, and to make decisions in the architectural views, if QS is not achieved. This process can take several iterations, and this is the whole cycle that we expect to accelerate and using ACCORDANT.

5. Evaluation with four BDA use cases

Our experimentation aims to compare development and deployment time for each iteration using ACCORDANT and other two frameworks reviewed in Section 3: 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 proposal in different domains through four use cases: UC1) Transport delay prediction, UC2) Near mid-air collision detection, UC3) Near mid-air collision risk analysis, and UC4) El Nino/Southern Oscillation cycles. Table 2 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. 5 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,[4] and the use cases description will be presented below.

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

Table 2
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 Nino/Southern oscillation	Weather	Polynomial regression	Batch	Deadline

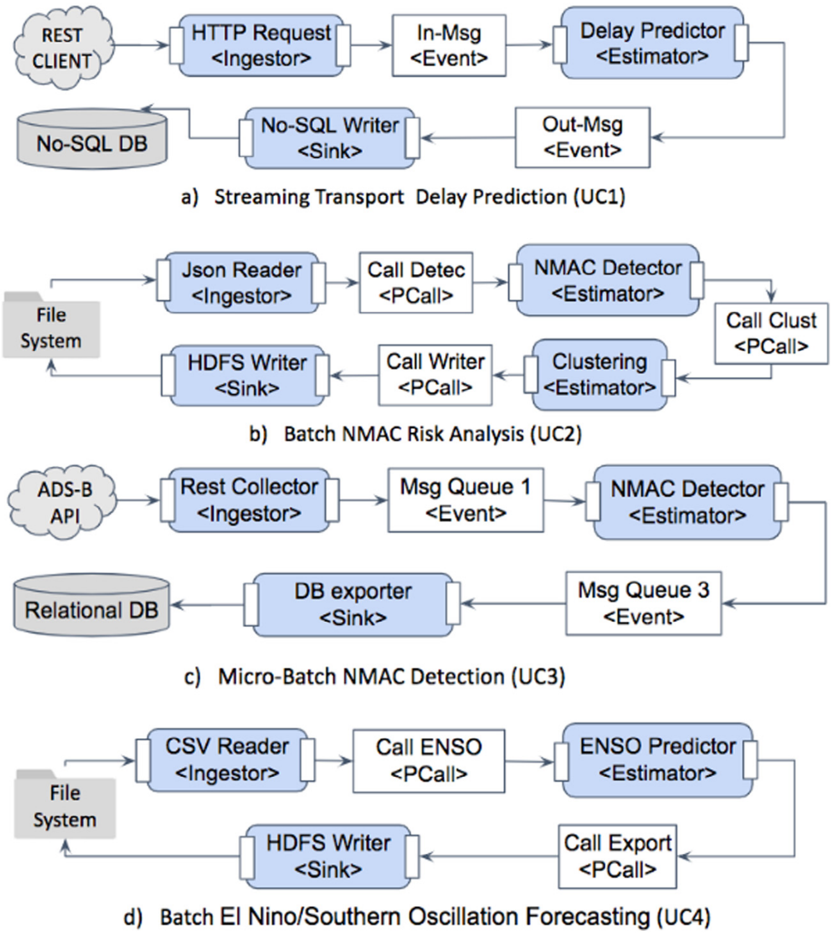


Fig. 5. Component diagrams of Use Cases.

5.1. Use case 1 (UC1)

The first use case was presented in Castellanos et al. (2018), and it deals with delay prediction of public transportation in Vancouver. Bus trips 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. 5a, starts with an ingestor component which receives HTTP request and put 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 productive environment as a delay predictor service, using Open-Scoring, and Kafka message broker, and MongoDB writer as target technologies. The QS were defined in terms of performance and modifiability attributes. The QS specifies that users make 1000 requests to delay prediction service under operations without load, and the responses must have an average latency lower than 2 s. Second QS states that when data scientist produces a new version of the predictive model (new PMML file), it must be updated at runtime within 10 s.

5.2. 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 are monitored. This use case is described in Fig. 5(b), and it was presented in Castellanos et al. (2019). NMAC detection comprises a pairwise comparison of flights: C_n^2), where n is the number of flights. Each comparison requires to calculate 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 JSON reader component. This ingestor calls 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 three air space ranges in nautical miles (nmi): 2 nmi, 20 nmi, 200 nmi, and 1500 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 (1500 nmi) is 1.4 GB of JSON files. This use case did not have real-time requirements due to its heavy workload nature, and therefore a performance QS for deadlines lower than one hour was defined.

5.3. 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. 5(c). The ingestor component consumed data through direct 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 are the same, since its inputs, outputs, and behavior are identical, so we can reuse such functional component definition, in spite of 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.

5.4. Use case 4 (UC4)

In this last use case, we used a public available data and PMML model (polynomial regression) of El Nino/Southern Oscillation (ENSO)[5] to implement a batch oriented pipeline, see Fig. 5d. The El Nino/Southern Oscillation (ENSO) cycle, was the strongest of the century which produced many problems throughout the world affecting South and North America countries with destructive flooding in some areas and strong drought in other areas. Data for this use case contains oceanographic and surface meteorological readings (geolocation, humidity, surface winds, sea surface temperatures, and subsurface temperatures) are taken from a series of buoys positioned throughout the equatorial Pacific. This data is expected to 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 a 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 min.

5.5. 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 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. These phases are performed iteratively, since in each iteration some improvements and refinements are done until the QS are achieved. Therefore, we measure the time invested in each iteration, and also we calculate the gain factor $GF(uc,f)$, as a metric to estimate the cumulative average of time reduction ratio for a use case uc, using framework f over I iterations. $GF(uc,f)$ is defined as follows:

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

We define the gain factor as a form to measure the incremental improving of using a high level abstractions to modify or refine an application until achieve an expected QS. The time for each use case, phase, and iteration was collected from two development teams which learnt and used the three frameworks to develop and deploy two use cases each one, while they were recording the time spent. The development and deployment process using ACCORDANT will be illustrated with UC4 in the next Section.

6. Experimentation

To design, develop, and deploy the four use cases, we followed ACCORDANT process detailed previously in Fig. 1. 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 Castellanos et al. (2018, 2019). The ACCORDANT projects are available in a public repository[6] as well as use cases and results.[7]

6.1. Definition of quality scenarios

QS are 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. 6 details architectural inputs of UC4 expressed using the ACCORDANT's input package DSL. The predictor component is required to have a deadline lower than 1 h in the QS *UC4_QS1*. Analyzing this QS, a sensitivity point (*UC4_SP1*) is identified to achieve the deadline metric by applying two tactics: *introduce concurrency* and *increase*. available resources. These tactics will be materialized in the software architecture design.

6.2. Development of data transformations and analytics model

The analytics model is trained and evaluated by the data scientist outside the ACCORDANT framework, and the resulting models were exported to PMML file to be loaded in the ACCORDANT functional model. In this case, the polynomial regression model of ENSO is downloaded and used. Fig. 7 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.

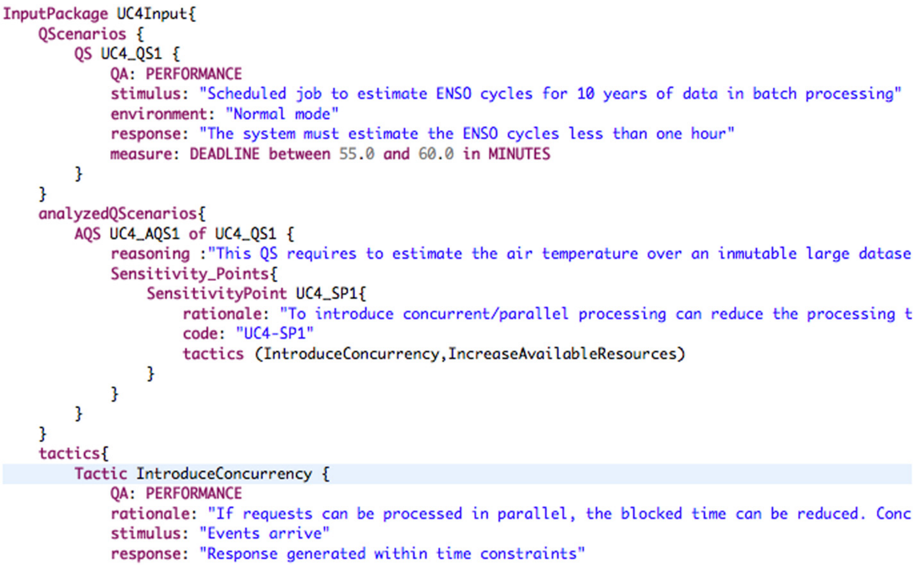
6.3. Design of software architecture — Functional view

FV models were designed using ACCORDANT Functional DSL to specify a component-connector structure for each use case. Two iterations of functional model were designed for UC4, and the last iteration is depicted in Fig. 8a. Since architectural inputs are required in this design, this package is imported using the keyword *use inputPackage*. The functional model specifies three components: (*CSVReader::Ingestor*, *ENSOPredictor::Estimator*,

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

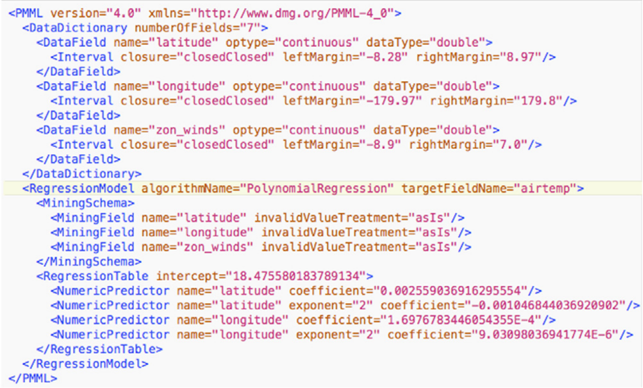
[6] http://github.com/kmilo-castellanos/accordant

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



```
InputPackage UC4Input{
    QScenarios {
        QS UC4_QS1 {
            QA: PERFORMANCE
            stimulus: "Scheduled job to estimate ENSO cycles for 10 years of data in batch processing"
            environment: "Normal mode"
            response: "The system must estimate the ENSO cycles less than one hour"
            measure: DEADLINE between 55.0 and 60.0 in MINUTES
        }
    }
    analyzedQScenarios{
        AQS UC4_AQS1 of UC4_QS1 {
            reasoning :"This QS requires to estimate the air temperature over an inmutable large datase
            Sensitivity_Points{
                SensitivityPoint UC4_SP1{
                    rationale: "To introduce concurrent/parallel processing can reduce the processing t
                    code: "UC4-SP1"
                    tactics (IntroduceConcurrency,IncreaseAvailableResources)
                }
            }
        }
    }
    tactics{
        Tactic IntroduceConcurrency {
            QA: PERFORMANCE
            rationale: "If requests can be processed in parallel, the blocked time can be reduced. Conc
            stimulus: "Events arrive"
            response: "Response generated within time constraints"
```

Fig. 6. Excerpt of input package models of UC4 using ACCORDANT DSLs.



```
<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>
```

Fig. 7. Excerpt of ENSO polynomial regression model of UC4 in PMML format.

and *HDFSWriter*::Sink), and two connectors: procedure calls *CallEnso::ProcCall* and *CallExport::ProcCall* which connect the components through ports. The components also include some properties such as connections and formats. Additionally, *ENSOPredictor* uses batch processing model, it has associated the PMML "ElNinoPolReg.pmml", obtained in the previous step, to provide the predictive behavior. The sensitivity point *UC4_SP1* aligns the architectural input (QS and tactics explained in Section 6.1) to ENSOPredictor. 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.

6.4. Design of software architecture — Deployment view

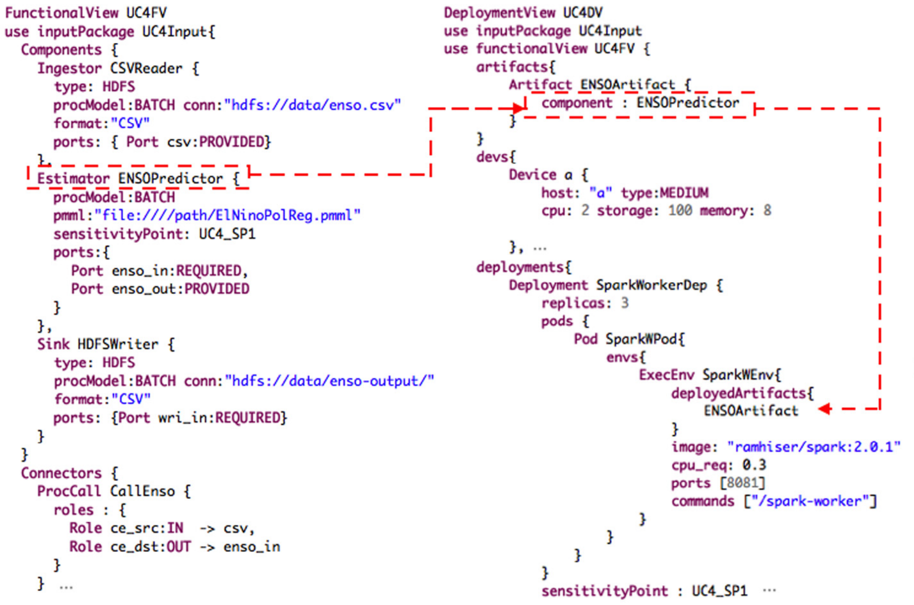
The deployment view models were designed using ACCORDANT DSL for each use case defined in the functional models. The UC4 deployment model had three iterations, and Fig. 8b details the last version. Given that DV is based on input package and functional view, they are imported by means of keyword *use inputPackage* and *functionalView* respectively. This view includes the artifacts that map connectors and components from functional view (e.g. *ENSOPredictor*) to deployable elements (e.g. *ENSOArtifact*). Devices and deployments were specified to support the computation requirements. For instance, deployments of Spark master and worker nodes (e.g. *SparkWorkerDep*) details the number of replicas, pods, and execution environments (*ExecEnv*). ExecEnv defines the docker image, CPU and memory requirements, ports, and commands along with the artifacts to be deployed (*ENSOArtifact*). Finally, the sensitivity point *UC4_SP1* associates the deployment SparkWorkerDep to performance QS, and the tactic *increase available resources* (see Section 6.1) to support distributed computing over a Spark cluster.

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 1, where *ENSOPredictor* component implements the PMML model in Spark technology. The Spark program defines data input and output from the Data Dictionary and Mining Schema embedded in PMML specifications. On the other hand, infrastructure code is the configuration files which specify the provision and configuration policies of Kubernetes cluster. Listing 2 shows an example of generated Kubernetes files. The whole code of use cases is publicly available in the accordant-usecases repository.

Listing 1: Generated Java Code of EnsoEstimator Component for Spark Streaming

```
SparkSession sparkSession = new SparkSession(sc.sc());
InputStream pmmlFile = new URL("file:////path/ElNinoPolReg.pmml")
EvaluatorBuilder b = new LoadingModelEvaluatorBuilder().load(pmmlFile);
Evaluator eval = builder.build();
TransformerBuilder pmmlTransformerBuilder =
new TransformerBuilder(evaluator)
    .withTargetCols().exploded(true);
List<StructField> fields = new ArrayList<StructField>();
fields.add(DataTypes.createStructField("latitude", DataTypes.DoubleType, true));
...
fields.add(DataTypes.createStructField("s_s_temp",DataTypes.DoubleType, true));
StructType schema = DataTypes.createStructType(fields);
Transformer pmmlTransformer = pmmlTransformerBuilder.build();
Dataset<Row> inputDs = sparkSession.read().schema(schema).csv("data/Elnino.csv");
TransformerBuilder tb = new TransformerBuilder(eval);
Transformer transformer = tb.build();
Dataset<Row> resultDs = transformer.transform(inputDs);
resultDs.write().option("header", "true").csv("/enso output/");
...
```



```
FunctionalView UC4FV
use inputPackage UC4Input{
  Components {
    Ingestor CSVReader {
      type: HDFS
      procModel:BATCH conn:"hdfs://data/enso.csv"
      format:"CSV"
      ports: { Port csv:PROVIDED}
    },
    Estimator ENSOPredictor {
      procModel:BATCH
      pmml:"file:////path/ElNinoPolReg.pmml"
      sensitivityPoint: UC4_SP1
      ports:{
        Port enso_in:REQUIRED,
        Port enso_out:PROVIDED
      }
    },
    Sink HDFSWriter {
      type: HDFS
      procModel:BATCH conn:"hdfs://data/enso-output/"
      format:"CSV"
      ports: {Port wri_in:REQUIRED}
    }
  }
  Connectors {
    ProcCall CallEnso {
      roles : {
        Role ce_src:IN  -> csv,
        Role ce_dst:OUT -> enso_in
      }
    } ...
```

```
DeploymentView UC4DV
use inputPackage UC4Input
use functionalView UC4FV {
  artifacts{
    Artifact ENSOArtifact {
      component : ENSOPredictor
    }
  }
  devs{
    Device a {
      host: "a" type:MEDIUM
      cpu: 2 storage: 100 memory: 8
    }, ...
  deployments{
    Deployment SparkWorkerDep {
      replicas: 3
      pods {
        Pod SparkWPod{
          envs{
            ExecEnv SparkWEnv{
              deployedArtifacts{
                ENSOArtifact
              }
              image: "ramhiser/spark:2.0.1"
              cpu_req: 0.3
              ports [8081]
              commands ["/spark-worker"]
            }
          }
        }
      }
    sensitivityPoint : UC4_SP1 ...
```

Fig. 8. Excerpt of functional (a) and deployment (b) models of UC4 using ACCORDANT DSLs.

Listing 2: Generated YAML Code from Deployment Specification for Kubernetes (Extract)

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: SparkWorkerDep
spec:
  replicas: 3
    spec:
      containers:
        name: SparkWEnv
        image: ramhiser/spark:2.0.1
        command: [/spark worker]
        ports:
          containerPort: 8081
        resources:
          requests:
            cpu: 0.3
...
```

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.

6.7. Solution monitoring

Performance metrics for each use case in operation were collected and validated against QS defined in Section 6.1. As a result, different deployment configurations were designed, deployed, and monitored in each iteration to observe the fulfillment of QS.

7. Results and discussion

Revisiting the related work reviewed in Section 3, we have shown in practice how ACCORDANT bridge the gap among analytics, software architecture, and DevOps. As presented in Table 1, ACCORDANT follows the SoC principle by means of three different languages to specify domain concerns. Analytics models in ACCORDANT are cross-industry and technology-neutral. In terms of software architecture, ACCORDANT supports QS specifications aligned to FV and DV, and these models can be specified independently but in an integrated way. Architectural tactics enable software architects to describe and communicate their decisions. Code generators offer flexibility and impact positively the development and deployment efficiency. Respecting DevOps practice, deployment models allow us to design deployment diagrams, not limited to a number of instances. Continuous deployment is supported via IaC and code generation, and QS-monitoring is implemented by injecting logging code in the generated applications. Finally, self-adaptation is not covered in the current version of ACCORDANT.

To summarize, though a large variety of component-connector metamodels have been previously proposed, as far as we know, our contribution resides in specialize a component-connector metamodel in the BDA domain, and integrate it with architectural inputs and deployment models to offer a holistic design. Additionally, this section presents and discusses the experimental results obtained during the iterative development and deployment phases of UC1, UC2, UC3, and UC4.

7.1. Development and deployment time

Fig. 9 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 inputs, 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 Inputs 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 biggest 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 time of development

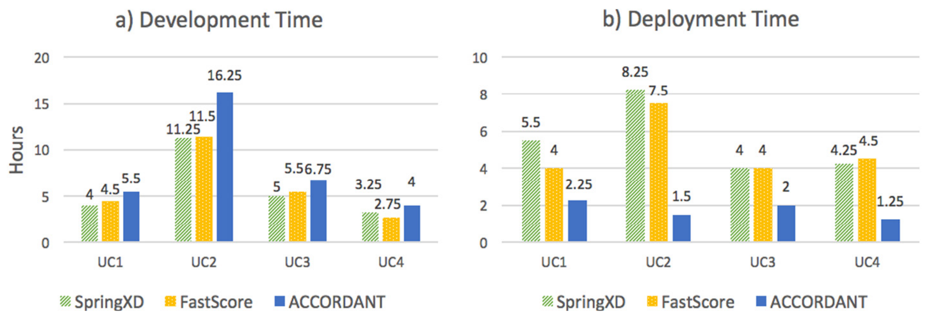


Fig. 9. Development and deployment time for use case.

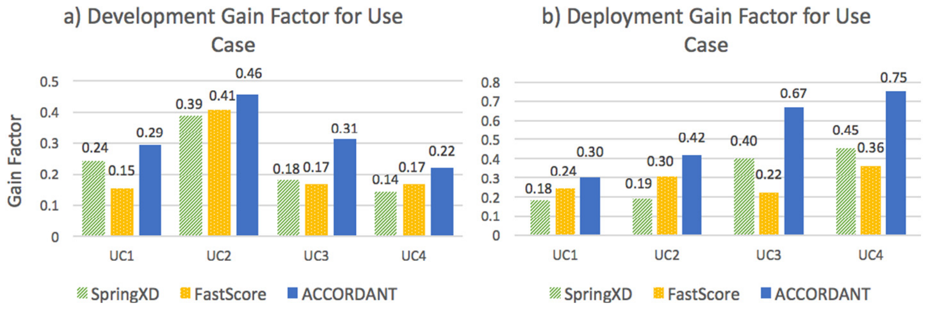


Fig. 10. Gain factor for use case.

as shown the marked decreasing between use cases, and the growing gain factor among iterations detailed in Fig. 9. These results suggest that ACCORDANT is most suitable for application involving multiple iterations, or in subsequent applications where reusing architectural decisions, models, and metrics can reduce development times.

7.2. Gain factor comparison

The gain factor metric presented in Eq. (1) in Section 5.5 was calculated for each use case and iteration of development and deployment phases as depicted in Fig. 10. ACCORDANT's gain factor was higher for all use cases, in the development phase (Fig. 10a), what 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, it means reducing in 46% the development time between consecutive iterations. The greatest gain factor difference over the other approaches was 0.13 in the UC3. Regarding the deployment gain factor (Fig. 10b), ACCORDANT also exhibited the highest gain factor, on an even higher proportion, up to 0.75 in UC4. This means each deployment iteration reduces the time in 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 of the IaC generation is not present in the other approaches.

8. Conclusions

We have presented a DevOps and DSM proposal to design, deploy, and monitor BDA solutions. We have positioned the ACCORDANT contributions within the related work. Four use cases from different domains were used to evaluate our approach against two BDA frameworks. As a result, ACCORDANT has shown to facilitate and accelerate iterative development and deployment phases by offering an integrated and high-level design BDA applications. The greatest time reduction was reported in the deployment phase, achieving up to 81% compared to other approaches. In contrast, the development times offered by ACCORDANT were greater. Despite the longer development time, deployment time is significantly reduced thanks to the QS, FV, and DV alignment. ACCORDANT's gain factor was higher, which implies a higher reduction time in each iteration.

In contrast, some limitations have emerged from experimentation. The development phase is slower than the other approaches for multiple reasons. The current version of the ACCORDANT's prototype requires supplementary manual coding what increases the development time. ACCORDANT also requires more design details and architectural inputs. These additional definitions are rewarded in consecutive iterations, so ACCORDANT is most suitable for application involving multiple iterations. Finally, our approach takes advantage of reusing architectural decisions and models, hence, first-time or one-time applications may not be benefited from our proposal.

As future work, the performance metrics collected along with FV and DV models could allow us to propose a performance model to predict the expected application-specific behavior based on the functional model, deployment model, and target technology to recommend optimal architecture configuration for a defined QS. Furthermore, we could include features to simulate and verify correctness properties over the models such as technology selection in the FV model and resource allocation in the DV model. Given that PMML provides a model verification schema to validate results accuracy, a future extension could incorporate automated model verification. This approach has been used for deploying analytics components and connectors on virtual machines over cloud infrastructure, but different paradigms such as serverless or fog computing may open new research lines.

Acknowledgments

This research is supported by Fulbright Colombia and the Center of Excellence and Appropriation in Big Data and Data Analytics (CAOBA), supported by the Ministry of Information Technologies and Telecommunications of the Republic of Colombia (MinTIC) through the Colombian Administrative Department

Table A.3
Abbreviations.

ADD	Attribute driven design method
ADS-B	Automatic dependent surveillance—broadcast
BDA	Big data analytics
DSL	Domain-specific language
DSM	Domain-specific model
DV	Deployment view
ENSO	El Nino/Southern oscillation
FV	Functional view
IaC	Infrastructure as code
NMAC	Near mid-air collision
PFA	Portable format for analytics
PMML	Predictive model markup language
QS	Quality scenario

of Science, Technology, and Innovation (COLCIENCIAS) within contract No. FP44842-anexo46-2015. The authors would like to thank an Amazon Web Services Educate for the student grant.

Appendix A. Abbreviations

See Table A.3.

References

Alrifai, M., Eichelberger, H., Qui, C., Sizonenko, R., Burkhard, S., Chrysos, G., 2014. Quality-Aware Processing Pipeline Modeling. Tech. Rep., QualiMaster Project.

Anandan, S., Bogoevici, M., Renfro, G., Gopinathan, I., Peralta, P., 2015. 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. ACM Press, New York, New York, USA, pp. 217–225. http://dx.doi.org/10.1145/2675743.2771879.

Artac, M., Borovsak, T., Di Nitto, E., Guerriero, M., Perez-Palacin, D., Tamburri, D.A., 2018. Infrastructure-as-code for data-intensive architectures: A model-driven development approach. In: 2018 IEEE International Conference on Software Architecture, ICSA. IEEE, pp. 156–165. http://dx.doi.org/10.1109/ICSA.2018.00025.

Bass, L., Weber, I., Zhu, L., 2015. DevOps: A Software Architect's Perspective. Addison-Wesley Professional.

Castellanos, C., Correal, D., Rodriguez, J.-D., 2018. Executing architectural models for big data analytics. In: Cuesta, C.E., Garlan, D., Pérez, J. (Eds.), Software Architecture. Springer International Publishing, Cham, pp. 364–371.

Castellanos, C., Pérez, B., Varela, C.A., Villamil, M.d.P., Correal, D., 2019. A survey on big data analytics solutions deployment. In: Bures, T., Duchien, L., Inverardi, P. (Eds.), Software Architecture. Springer International Publishing, Cham, pp. 195–210.

Cervantes, H., Kazman, R., 2016. Designing Software Architectures: A Practical Approach. Addison-Wesley Professional.

Chen, H.-M., Kazman, R., Haziyev, S., 2016. Agile big data analytics for web-based systems: An architecture-centric approach. IEEE Trans. Big Data 2 (3), 234–248. http://dx.doi.org/10.1109/TBDATA.2016.2564982.

Chen, H.-M., Schütz, R., Kazman, R., Matthes, F., 2017. How Lufthansa capitalized on big data for business model renovation. MIS Q. Executive 1615 (14), 299–320.

CloverDX, CloverDX Data Integration. URL https://www.cloverdx.com/.

Erl, T., Khattak, W., Buhler, P., 2016. Big Data Fundamentals: Concepts, Drivers & Techniques. Prentice Hall Press, URL http://dl.acm.org/citation.cfm?id=2898954.

Gorton, I., Klein, J., 2014. Distribution, data, deployment: Software architecture convergence in big data systems. IEEE Softw. 32 (3), 78–85.

Gribaudo, M., Iacono, M., Kiran, M., 2017. A performance modeling framework for lambda architecture based applications. Future Gener. Comput. Syst. http://dx.doi.org/10.1016/j.future.2017.07.033.

Guerriero, M., Tajfar, S., Tamburri, D., Di Nitto, E., 2016. Towards a model-driven design tool for big data architectures. In: 2nd IWBDSE. IEEE, pp. 37–43.

Huang, Y., Lan, X., Chen, X., Guo, W., 2015. Towards model based approach to hadoop deployment and configuration. In: 12th WISA. IEEE, pp. 79–84. http://dx.doi.org/10.1109/WISA.2015.65.

Lechevalier, D., Ak, R., Lee, Y.T., Hudak, S., Foufou, S., 2015. A neural network meta-model and its application for manufacturing. In: 2015 IEEE International Conference on Big Data. pp. 1428–1435.

Marz, N., Warren, J.J.O., 2015. Big Data: Principles and Best Practices of Scalable Real-Time Data Systems. URL https://www.manning.com/books/big-data.

Open Data Group, FastScore. URL https://www.opendatagroup.com/fastscore.

Perez-Palacin, D., Merseguer, J., Requeno, J.I., Guerriero, M., Di Nitto, E., Tamburri, D.A., 2019. A UML profile for the design, quality assessment and deployment of data-intensive applications. Softw. Syst. Model. 18 (6), 3577–3614.

Ranjan, R., 2014. Streaming big data processing in datacenter clouds. IEEE Cloud Comput. 78–83, URL http://mahout.apache.org.

Rexer, K., 2013. 2013 Data Miner Survey. Tech. Rep., Rexer Analytics.

Rexer, K., Gearan, P., Allen, H., 2016. 2015 Data Science Survey. Tech. Rep., Rexer Analytics.

Rozanski, N., Woods, E., 2005. Software Systems Architecture: Working with Stakeholders using Viewpoints and Perspectives. Addison-Wesley.

Sujeeth, A.K., Lee, H., Brown, K.J., Chafi, H., Wu, M., Atreya, A.R., Olukotun, K., Rompf, T., Odersky, M., 2011. OptiML: An implicitly parallel domain-specific language for machine learning. In: 28th ICML. pp. 609–616.

Taylor, R.N., Medvidovic, N., Dashofy, E.M., 2010. Software architecture: Foundations, theory and practice. John Wiley and Sons, Inc.

Ullah, F., Babar, M.A., 2019. Architectural tactics for big data cybersecurity analytics systems: A review. J. Syst. Softw. 151, 81–118.

Wegener, D., Rüping, S., 2010. On reusing data mining in business processes-a pattern-based approach. In: International Conference on Business Process Management. Springer, pp. 264–276.

Wojcik, R., Bachmann, F., Bass, L., Clements, P., Merson, P., Nord, R., Wood, B., 2006. Attribute-Driven Design (ADD), Version 2.0. Tech. Rep., Carnegie-Mellon Univ Pittsburgh PA Software Engineering Inst.

Cristian Castellanos is a Ph.D. candidate at the Department of Systems and Computing Engineering, Universidad de Los Andes, Colombia. His research tackles the deployment challenges of big data analytics solutions using model-driven architecture motivated by real-life experiences. He holds an M.Sc. in Systems and Computing Engineering (cum laude) from Universidad de Los Andes focused on enterprise architecture alignment between business and information domain applying model-driven engineering and ontology matching. Since 2016, he has been working at Alianza Caoba, a public–private initiative to gather Colombian government, industry and scholar community around applied research on big data analytics. In 2019, he was awarded a Fulbright scholarship for a research stay at RPI, NY to study the deployment of big data analytics applications in avionics.

Dr. Carlos A. Varela received his B.S. with honors, M.S., and Ph.D. in Computer Science at the University of Illinois at Urbana-Champaign. Dr. Varela is Associate Editor and Information Director of the ACM Computing Surveys journal, and has served as Guest Editor of the Scientific Programming journal. Dr. Varela is a recipient of several research grants including the NSF CAREER award, two IBM SUR awards, and two IBM Innovation awards. His current research interests include web-based and internet-based computing, middleware for adaptive distributed systems, concurrent programming models and languages, and software development environments and tools. For more information on Prof. Varela's group's research, please visit the Worldwide Computing Lab at http://wcl.cs.rpi.edu/.

Dr. Darío Correal is an Associate Professor of the Department of Systems and Computing at the University of Los Andes. Darío Correal is a Systems Engineer from the Universidad de Los Andes, with a Master's Degree in Systems Engineering and a Doctorate in Engineering from the Universidad de Los Andes. His specific research interests are Software Architecture, Solution Architecture, and Self-Adaptable Architectures. His research carried out within the TICsW groups. Dr. Correal is a recipient of awards including third place of ACM-SIGPLAN Student Research Competition - OOPSLA'06, and third place of Student's Competition - I2LOR' 06. In the industry, he has over ten years of experience in software development and management of development teams.