A Model-Driven Architectural Design Method for Big Data Analytics Applications

Camilo Castellanos*, Boris Pérez*†, Darío Correal*
*System Engineering and Computing Department
University of Los Andes, Bogotá, Colombia
Email: cc.castellanos87, br.perez41, dcorreal@uniandes.edu.co
†Department of Systems
Francisco de Paula Santander University, Cúcuta, Colombia

Carlos A. Varela
Computer Science Department
Rensselaer Polytechnic Institute, Troy, NY, USA
Email:cvarela@cs.rpi.edu

***Abstract*—Big data analytics (BDA) applications use machine learning to extract valuable insights from large, fast, and heterogeneous data sources. The architectural design and evaluation of BDA applications entail new challenges to integrate emerging machine learning algorithms with cutting-edge practices whilst ensuring performance levels even in the presence of large data volume, velocity, and variety (3Vs). This paper presents a design process approach based on the Attribute-Driven Design (ADD) method and Architecture tradeoff analysis method (ATAM) to specify, deploy, and monitor performance metrics in BDA applications supported by domain-specific modeling and DevOps. Our design process starts with the definition of architectural drivers, followed by functional and deployment specification through integrated high-level modeling which enables quality scenarios monitoring. We used two use cases from avionics to evaluate this proposal, and the preliminary results suggest advantages by integrating multiple views, automating deployment and monitoring compared to similar approaches.**

***Index Terms*—Software architecture, Attribute-Driven Design, ADD, ATAM, Big data analytics deployment, DevOps, Domain-specific model, Quality Scenarios**

I. 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*) while maintaining expected performance. BDA development involves three knowledge domains: business, analytics, and technology. In the business domain, business users define business goals and quality scenarios (QS) to drive analytics projects. In the analytics domain, business goals are translated into specific analytics tasks by data scientists. In the technology domain, architects make decisions in terms of tactics, patterns, and deployment strategies addressing QS. The current design approaches do not address this multi-domain nature and complexity involved in BDA application development which frequently leads to delayed deployments [1]. Due to the lack of methods and tools to enable integration and alignment of multiple domains, BDA development presents a costly transition between development and production environments ("Deployment Gap" phenomenon [1]).

ACCORDANT [2] is a Domain-Specific Model (DSM) approach to formally specify, develop, deploy, and monitor BDA solutions bridging the gap between analytics and IT domains. This paper proposes an extension of the ACCORDANT Method by including architectural inputs (drivers) and aligning to the Attribute-Driven Design Method [3] (ADD 3.0), and to promote the architecture testability following evaluation methods such as ATAM (Architecture tradeoff analysis method) [4]. The proposed method is a model-driven approach that allows us to design, assess, and deploy integrated BDA applications based on architectural drivers: quality scenarios, constraints, tactics and sensitivity points. This proposal was validated with two use cases from the avionics field by designing functional and deployment models, and assessing performance QS in distributed batch and micro-batch processing contexts. The contributions of this paper are: 1) A DSM method to design and evaluate BDA architectures aligned to drivers thus accelerating iterative development and deployment. 2) Three integrated domain-specific languages (DSLs) to specify architectural inputs, functional and deployment view. 3) The experimentation of this proposal on two avionics use cases using different deployment strategies and QS.

The rest of this paper is organized as follows. In Section II describes the background. Section III reviews related work. Section IV details our proposal. Section V describes the experimentation. Section VI reports preliminary results. Finally, Section VII summarizes the conclusions and next steps.

II. Background

A. 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*. The architectural design is driven by QS and functional requirements through a systematic design method, such as ADD [3]), and it could be evaluated using methods such as ATAM [4]. ADD comprises 7 steps: 1) Review inputs (purpose, functional requirements, QS, and constraints). 2) In each ADD iteration, a design goal is defined from these

The authors would like to thank Amazon Web Services educational research for granting us their cloud resources.

978-1-7281-4659-1/20/$31.00 ©2020 IEEE
DOI 10.1109/ICSA-C50368.2020.00026

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. and 7) Analyze current design and review goal achievement and design purpose, and start a new iteration (from step 2), if selected drivers are not satisfied.

B. Infrastructure as Code and BDA Deployment

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 environments, operative systems, middleware, configuration resources and allocate them automatically. Portability plays a key role to deploy, operate, and evolve BDA applications due to the wide range of BDA technologies. Hence, portable standards appear such as Predictive Model Markup Language (PMML)[1]. PMML models specify machine learning models and data transformations along with their metadata. The PMML standard is supported by a wide range of data science tools such as R, SAS, IBM SPSS, among others.

III. Related Work

Several works have proposed frameworks to build and deploy BDA applications. We review and compare some of the most relevant works in Table I highlighting the important features. In the analytics domain, we compare if they use separation of concerns (*SoC*), cross-industry application (*CI*), and support of technology-neutral models (*TNM*). Regarding software architecture concepts, we include: QS specification (*QSS*), functional (*FV*) and deployment (*DV*) views, tactics (*AT*), and target-technology assignment (*TTA*: predefined technologies (*P*) or extensible code generators (*C*). Considering DevOps practices, deployment specification (*DS*) defines if only a number of instances (*I*) per component or a whole deployment diagram (*D*) can be described. Finally, practices as continuous deployment (*CD*), QS monitoring (*QSM*), and self-adaptation (*SA*) support IT operations.

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. [5] introduce a DSM framework for predictive analytics of manufacturing data using artificial neural networks to generate analytics models. Sujeeth et al. present in [8] OptiML, a DSL for machine learning which describes analytics functions using a statistical model that covers a subset of ML algorithms, this analytics functions are analyzed and optimized before the code generation.

In contrast, we found another group of studies interested in infrastructure concerns of BDA applications leaving aside their functional components. Gribaudo et al. [6] propose a modeling framework based on graph-based language to evaluate the system's performance of running applications that follow the lambda architecture pattern. Huang et al. [7] introduce a model to design, deploy, and configure Hadoop clusters through architecture metamodel and rules, which describe BDA infrastructure and deploy automation.

A final group of works combines functional definitions and deployment specifications. QualiMaster [9] 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. Fastscore [10] is a commercial framework to design and deploy analytics models. Analytics components are conventionally developed using a determined programming language or technology-neutral models, and once imported to the platform, they can be connected to data inputs and outputs. SpringXD [11] is a unified, distributed, and extensible system for data ingestion, analytics, processing, and export to simplify BDA development and deployment. Finally, the DICE project in [12] presents a DSM offering big data design that comprises data, computation, technology-frameworks, and deployment concepts to design and deploy data-intensive applications. DICE proposes a model-driven approach to develop application models that are automatically transformed into IaC.

IV. The ACCORDANT Method

This proposal aims at offering a high-level approach to design BDA solutions starting from architectural artifacts, instead of source code. Specifically, we propose an architecture design and development method based on ACCORDANT [2] framework to deal with architectural drivers, functional, and deployment views. Our proposal comprises a design and deployment method, and its underlying metamodel. This metamodel extends that proposed in [2] by including architectural inputs and serverless deployments. Fig. 1 depicts the ACCORDANT Method steps, which specializes and integrates ADD and ATAM concepts in the BDA domain.

The steps performed in the ACCORDANT framework are framed in solid lines, while the steps made with external tools are in dotted lines. ACCORDANT is iterative and composed of seven steps: 1) Elicitation of drivers (business goals, QS, and constraints) by business users and architects. 2) The data scientist builds and data transformations and analytics models (exported as PMML files) addressing the business goals. 3) The architect designs the software architecture in terms of *functional view*(FV) and *deployment view*(DV). FV makes use of PMML models to specify the analytics components' 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 is executed to provision infrastructure and install the software. 7) QS are monitored in operation, and new design iterations can be made to fulfill the drivers.

A. Architectural Drivers Elicitation

According to ADD and ATAM, architecture design and evaluation are driven by predefined quality scenarios (QS)

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

TABLE I
RELATED WORK

Work	SoC	Business (Analytics)		Software Architecture					DevOps			
		CI	TNM	QSS	FV	DV	AT	TTA	DS	CD	QSM	SA
Lechevalier et al. [5]		✓	✓		✓							
Gribaudo et al. [6], Huang et al. [7]		✓		✓					D		✓	
OptiML [8]		✓	✓		✓			C		✓		
Qualimaster [9]	✓			✓	✓	✓	✓			✓	✓	✓
FastScore [10]	✓	✓	✓		✓			C	I	✓	✓	
SpringXD [11]	✓	✓	✓		✓			P	I	✓	✓	✓
DICE [12]	✓	✓		✓	✓	✓		C	D	✓	✓	✓
ACCORDANT	✓	✓	✓	✓	✓	✓	✓	C	D	✓	✓	

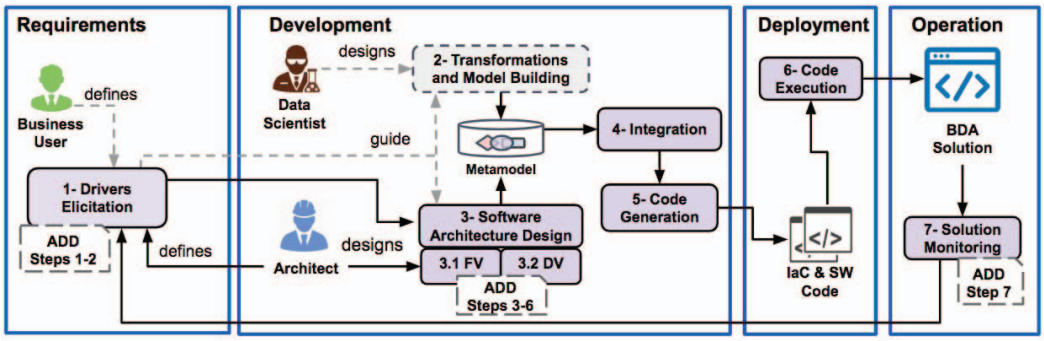

Fig. 1. ACCORDANT Method Overview

which must be achieved through design decisions compiled in well-known catalogs of architectural patterns and tactics. QS and tactics are inputs of the architecture design, therefore we include these initial building blocks in the ACCORDANT metamodel along with other concepts like constraints. Fig. 2 details the main input building blocks grouped by a (*Project*) which contains the elements required to start the architectural design: QS (*QScenario*), Analyzed QS (*AnalyzedQS*), *SentivityPoint* and *Tactic*. A *QScenario* determines a quality attribute requirement for a specific *Artifact*. Thus, for instance, a QS could be defined as "latency<=3 seconds for an artifact (software component or connector). A QS is analyzed through a *AnalyzedQS*, and sensitivity points. A *SensitivityPoint* is a decision's property (a set of elements and their relationships within architectural views) that is critical for achieving the QS, and that such decision is the application of a *Tactic* to a specific application context. Finally, *Constraints* restrict architectural decisions, e.g. mandated technologies, vendors, or processing models. This step covers ADD's steps 1 and 2.

B. Analytics Model Building

The data scientist build and evaluate data transformations and analytics models using data science tools, which are independent of ACCORDANT. This approach decouples analytics models and software architecture supported by the portability given by PMML format, but also it enables us to offer an integrated multi-domain framework.

C. Software Architecture Design

Once drivers are defined in step 1, architecture is designed in the *step 3* and expressed on the views instantiating tactics

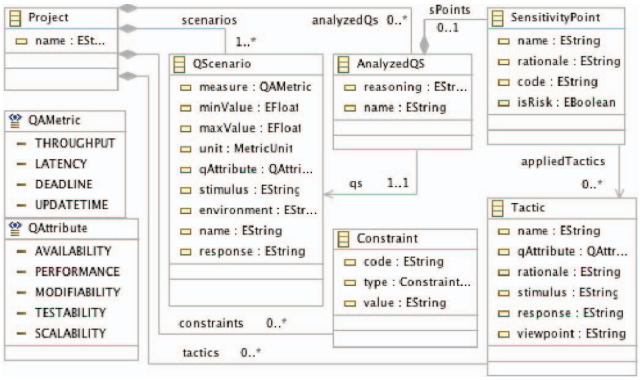

Fig. 2. Excerpt of Architectural Inputs Metamodel.

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. This step spans from steps 3 to 6 in ADD.

Functional View 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. FV is expressed in a component-connector model. Sensitivity points 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 predictive models and data transformers respectively. A *Component* exposes required and provided *Ports*. *Connectors* metaclasses transfer data or control flow among components through an input or output *Roles*. A set of connector types are defined: *Procedure Call*, *Event*, *Stream*, *Adaptor*, *Distributor* and *Arbitrator*.

Deployment Viewpoint includes DevOps practices starting with the specification of how software artifacts are deployed on a set of computation nodes. DV metamodel comprises *Pod*, *ExposedPort*, and *Deployment* metaclasses to operationalize BDA applications. A *FV* model can be deployed in different *DV* models either to use a different strategy or to test the fulfillment of predefined QS. 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. *Devices* (physical or virtual), *Pods*, and *ExecEnvironment*) constitute the main elements to provision virtual machines or containers-based infrastructures. On the other hand, *ServerlessEnv* element describes a computing environment in which the cloud provider dynamically manages the allocation of machine resources. Finally, *Artifacts* correspond to executable or deployable representations of functional elements (i.e. components and connectors from FV) which can be deployed on either execution or serverless environments.

D. Integration, Code Generation, and Execution

Once PMML, FV and DV models are designed and integrated, code generation takes place using 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 their properties and constraints. Such assignment enables us to generate code for target technology restricted to those constraints. The analytics model's inputs and outputs are transformed to the component's interfaces (required and provided respectively). To monitor QS, the code generators include specific machinery at application level to measure specific metrics (e.g. response time, throughput, deadline, etc) for each artifact according to its associated QS. This allows us to reduce code for logging starting from high-level quality specifications. On the IaC side, DV model is transformed into Kubernetes' configuration files, used to create and configure infrastructure over the Kubernetes where software artifacts can be automatically deployed using the FV-DV mappings.

E. Solution Monitoring

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. This process can take several iterations, and this is the whole cycle that we expect to accelerate and using ACCORDANT. This ACCORDANT's step corresponds to analyze drivers' achievement in ADD (step 7), and to analyze architectural approaches evaluated against each scenario in ATAM.

V. Experimentation with Avionics Use Cases

Our experimentation aims to compare development and deployment time for each iteration with other two frameworks reviewed in Section III: FastScore and SpringXD. We chose these frameworks because they are the closest to our approach, and they support portable analytics models.

We validated our proposal using two use cases: UC1) Near mid-air collision detection, and UC2) Near mid-air collision risk analysis. 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. More details about the use cases can be found in [13], and source code is publicly available[2].

Use case 1 (*UC1*) 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. NMAC detection comprises a pairwise comparison of flights to calculate location, speeds and heading to determine the risk level of NMAC. 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, and these results are stored back in the file system. This use case requires a heavy workload nature, and therefore a performance QS for deadlines lower than one hour was defined.

Use case 2 (*UC2*) is a real-time application to detect NMAC within an air space range. 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 UC1 and UC2 are the same, since its inputs, outputs, and behavior are identical, so we can reuse such functional component definition, though their deployments are different regarding the QS constraints. Given the near real-time nature of this application, latency is the critical QS.

A. Architectural Drivers Elicitation

The business goal is to group NMAC events to identify potential risky zones and times within specific air-spaces. A scheduled job to detect risky clusters is processed in batch every day. Fig 3 details drivers expressed using the ACCORDANT's DSL. The *NMACDetector* component is required to have a deadline lower than 1 hour in the QS *UC1_QS1*. Analyzing this QS, a sensitivity point (*UC1_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.

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

```
InputPackage UC1Inputs{
    QScenarios {
        QS UC1_QS1 {
            QA:PERFORMANCE
            stimulus: "Scheduled job for clustering of NMAC events"
            environment: "Normal mode"
            response: "The system must predict NMAC events in less than one hour"
            measure: DEADLINE between 55.0 and 60.0 in MINUTES
        }
    }
    analyzedQScenarios{
        AQS UC1_AQS1 of UC1_QS1   {
            reasoning :"This QS requires to assign NMAC events to risk clusters
                over inmutable large dataset in 60 minutes using batch processing"
            Sensitivity_Points{
                SensitivityPoint UC1_SP1{
                    rationale: "To introduce concurrent/parallel processing can
                        reduce the processing time by reducing blocked time. Paral
                    code: "UC1-SP1"
                    tactics (IntroduceConcurrency, IncreaseAvailableResources)
                }
```

Fig. 3. Excerpt of Input Package Models of UC1 Using ACCORDANT DSLs

```
FunctionalView UC1FVModel
use inputPackage UC1Inputs{
    Components {
        Ingestor JsonReader {
            type:HDFS ports: { Port adsb:PROVIDED}
            procModel:BATCH conn:"hdfs://ads-b/input/" format:"JSON"
        },
    A   Estimator NMACDetector {
            procModel:BATCH  pmml:"file://.../NMACTreeModel.pmml"
            sensitivityPoint: UC1_SP1
            ports:{Port nmac_in:REQUIRED, Port nmac_out:PROVIDED}
        },
        Estimator NMACClustering {
            procModel:BATCH  pmml:"file://.../NMACKmeans.pmml"
            ports:{Port clus_in:REQUIRED, Port clus_out:PROVIDED}
        },
    Connectors {
        ProcCall CallNMACDetector {
         roles : {
            Role cd_src:IN  -> adsb,
            Role cd_dst:OUT -> nmac_in
          }
        },
```

Fig. 4. Excerpt of Functional Models of UC1 Using ACCORDANT DSL

B. Data Transformations and Analytics Models

Analytics models were trained and evaluated by the data scientist using Scikit-learn, exported to PMML, and loaded in the ACCORDANT FV model. In this case, the decision tree and K-means models will be assigned in the FV specification.

C. Design of Software Architecture

FV models were designed using ACCORDANT Functional DSL to specify a component-connector structure for each use case, Fig. 4 depicts the UC1's FV model. Since drivers are required in FV, this package is imported using the keyword *use*. The FV model specified four components (*JsonReader*, *NMACDetector*, *NMACClustering*, and *HDFSWriter*), and three procedure call connectors: *CallNMACDetector*, *CallClustering*, and *CallWriter* which connect the components through ports. Additionally, *NMACDetector* uses batch processing model, and it has associated "NMACTreeModel.pmml" obtained in the previous step. The sensitivity point *UC1_SP1* aligns the drivers to the *NMACDetector* as part of the *introduce concurrency* tactic realization. *NMACDetector* will be translated into a distributed processing component which must be supported by the target technology.

DV models were designed using ACCORDANT DSL for UC1 defined in the FV, see Fig. 5. Given that DV is based

```
DeploymentView UC1DV
use inputPackage UC1Inputs use functionalView UC1FVModel {
  artifacts{
    Artifact ReaderArtifact {
        component : JsonReader sensitivityPoint : UC1_SP1
    },
B   Artifact NMACArtifact {component: NMACDetector   A
    },
  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{
                B NMACArtifact,ReaderArtifact,
                    ClusteringArtifact,WriterArtifact
                }
                image: "ramhiser/spark:2.0.1"
                cpu_req: 0.3
                ...
```

Fig. 5. Excerpt of Deployment Models of UC1 Using ACCORDANT DSL

on the input package and FV model, they are imported using the keyword *use*. This view includes the artifacts that map connectors and components from FV to deployable elements in DV. For instance, *NMACDetector* (see markers *A*) is mapped to *NMACArtifact*, and deployed in *SparkWEnv* (see markers *B*). Devices and deployments were specified to support the computation requirements. For instance, deployments of Spark master and worker nodes (e.g. *SparkWorkerDep*) details replicas, pods and execution environments (*ExecEnv*). ExecEnv defines the docker image, resources, and ports along with the artifacts to be deployed. Finally, the sensitivity point *UC1_SP1* associates the deployment SparkWorkerDep to performance QS, and the tactic *increase available resources* (see Section V-A) to support distributed computing over a Spark cluster.

D. Integration, Code Generation, and Execution

Once FV and DV models were designed and integrated, code generators produced functional code and IaC. The target technology selected was Apache Spark, so *NMACDetector* component implements the PMML model in a Spark driver program. The Spark program defines data input and output from the Data Dictionary and Mining Schema embedded in PMML specifications. On the other hand, the infrastructure code was generated as Kubernetes' configuration files. Kubernetes code was executed on the AWS cloud using Amazon Kubernetes and EC2 services. After that, the software code was installed over the cluster to operationalize the solution.

E. Solution Monitoring

Deadline and latency metrics for each use case were collected in operation and validated against QS defined in Section V-A. As a result, different deployment configurations were designed, deployed and monitored in each iteration to monitor the fulfillment of QS.

VI. Preliminary Results

Revisiting the related work reviewed in Section III, we have shown how the ACCORDANT Method fills some gaps

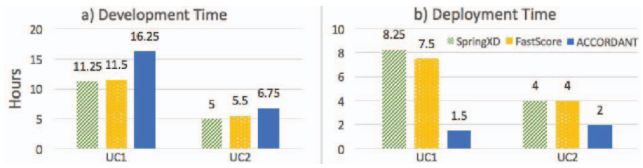

Fig. 6. Development and Deployment Time for Use Case

in BDA architecture. As presented in Fig. I, ACCORDANT follows the SoC principle using 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. Code generators promote flexibility and faster development and deployment. Respecting DevOps practice, deployment models allow us to design deployment diagrams and generate IaC to provision such resources semi-automatically. The solution monitoring is aligned to the initial QS specification and implemented by injecting logging code in the generated applications. Finally, self-adaptation is not covered in the current version.

Regarding the development and deployment effort, Fig. 6 depicts the average times invested for UC and two development teams. These teams developed the UCs using each framework and taking drivers (QS, constraints, and tactics) and the PMML model as input. Each UC was deployed to cloud containers, and the QS monitored using the features offered by each framework. The development time using ACCORDANT was higher (between 22.7% and 44.4%) compared to SpringXD and Fastscore, but the deployment time was significantly lower (between 50% and 81.8%) using ACCORDANT. The higher development time can be explained by the time required to specify architectural inputs and FV models. Besides, the current ACCORDANT prototype generates functional code for estimators, but ingestor, sinks, and connectors still require manual coding. Although ACCORDANT required more effort in the development phase, this effort was rewarded during the deployment phase, where infrastructure and QS-monitoring are provided automatically aligned to QS, unlike other approaches. The biggest time differences arose from UC1 that demanded more time because it included a more complex pipeline, involving two estimators. These results suggest ACCORDANT is more suitable for application involving multiple iterations, or in subsequent applications where reusing architectural elements can reduce development times.

VII. Conclusions

We have presented a design method to specify, deploy, and monitor BDA solutions. Two avionics use cases were used to evaluate our approach against two BDA frameworks. As a result, ACCORDANT has shown to facilitate and accelerate iterative deployment by offering an integrated and high-level design BDA applications by investing more effort in the design phase. 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 extra manual coding. 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.

The next steps include a model to predict the expected performance based on FV and DV models, target technologies, and collected metrics to recommend the optimal architecture configuration given a set of drivers. Furthermore, we are developing validation rules to check correctness properties against architectural constraints, e.g. technology conformance, resource availability, and architectural mismatch, taking advantage of the integration among drivers, FV and DV. Finally, the experimentation has been performed using containers in the DV, but we expect to include serverless and/or fog computing deployment which can open new challenges.

References

[1] H.-M. Chen, R. Schütz, R. Kazman, and F. Matthes, "How Lufthansa Capitalized on Big Data for Business Model Renovation," *MIS Quarterly Executive*, vol. 1615, no. 14, pp. 299–320, 2017.

[2] C. Castellanos, D. Correal, and J.-D. Rodriguez, "Executing Architectural Models for Big Data Analytics," in *Software Architecture*, C. E. Cuesta, D. Garlan, and J. Pérez, Eds. Cham: Springer International Publishing, 2018, pp. 364–371.

[3] H. Cervantes and R. Kazman, *Designing software architectures: a practical approach*. Addison-Wesley Professional, 2016.

[4] P. Clements, R. Kazman, M. Klein *et al.*, *Evaluating software architectures*. Tsinghua University Press Beijing, 2003.

[5] D. Lechevalier, R. Ak, Y. T. Lee, S. Hudak, and S. Foufou, "A Neural Network Meta-Model and its Application for Manufacturing," in *2015 IEEE International Conference on Big Data*, 2015, pp. 1428–1435.

[6] M. Gribaudo, M. Iacono, and M. Kiran, "A Performance Modeling Framework for Lambda Architecture Based Applications," *Future Generation Computer Systems*, jul 2017.

[7] Y. Huang, X. Lan, X. Chen, and W. Guo, "Towards Model Based Approach to Hadoop Deployment and Configuration," in *12th WISA*. IEEE, sep 2015, pp. 79–84.

[8] A. K. Sujeeth, H. Lee, K. J. Brown, H. Chafi, M. Wu, A. R. Atreya, K. Olukotun, T. Rompf, and M. Odersky, "OptiML: An Implicitly Parallel Domain-Specific Language for Machine Learning," in *28th ICML*, 2011, pp. 609—-616.

[9] M. Alrifai, H. Eichelberger, C. Qui, R. Sizonenko, S. Burkhard, and G. Chrysos, "Quality-aware Processing Pipeline Modeling," QualiMaster Project, Tech. Rep., 2014.

[10] Open Data Group, "FastScore." [Online]. Available: https://www.opendatagroup.com/fastscore

[11] S. Anandan, M. Bogoevici, G. Renfro, I. Gopinathan, and P. 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*. New York, New York, USA: ACM Press, 2015, pp. 217–225.

[12] M. Artac, T. Borovsak, E. Di Nitto, M. Guerriero, D. Perez-Palacin, and D. A. Tamburri, "Infrastructure-as-Code for Data-Intensive Architectures: A Model-Driven Development Approach," in *2018 IEEE International Conference on Software Architecture (ICSA)*. IEEE, apr 2018, pp. 156–165.

[13] C. Castellanos, B. Pérez, C. A. Varela, M. d. P. Villamil, and D. Correal, "A survey on big data analytics solutions deployment," in *Software Architecture*, T. Bures, L. Duchien, and P. Inverardi, Eds. Cham: Springer International Publishing, 2019, pp. 195–210.