Model-based Big Data Analytics-as-a-Service: Take Big Data to the Next Level

Claudio A. Ardagna, Valerio Bellandi, Michele Bezzi, Paolo Ceravolo, Ernesto Damiani, Cedric Hebert

Abstract—The Big Data revolution promises to build a data-driven ecosystem where better decisions are supported by enhanced analytics and data management. However, major hurdles still need to be overcome on the road that leads to commoditization and wide adoption of Big Data Analytics (BDA). Big Data complexity is the first factor hampering the full potential of BDA. The opacity and variety of Big Data technologies and computations, in fact, make BDA a failure prone and resource-intensive process, which requires a trial-and-error approach. This problem is even exacerbated by the fact that current solutions to Big Data application development take a bottom-up approach, where the last technology release drives application development. Selection of the best Big Data platform, as well as of the best pipeline to execute analytics, represents then a deal breaker. In this paper, we propose a *return to roots* by defining a Model-Driven Engineering (MDE) methodology that supports automation of BDA based on model specification. Our approach lets customers declare requirements to be achieved by an abstract Big Data platform and smart engines deploy the Big Data pipeline carrying out the analytics on a specific instance of such platform. Driven by customers' requirements, our methodology is based on an OWL-S ontology of Big Data services and on a compiler transforming OWL-S service compositions in workflows that can be directly executed on the selected platform. The proposal is experimentally evaluated in a real-world scenario focusing on the threat detection system of SAP.

Index Terms—Big Data, Model-Driven Architecture, OWL-S

1 Introduction

Big Data has become a major IT trend that involves academia, research institutions, and industries. According to IDC [1], "*revenues for Big Data and business analytics will grow from \$150.8 billion in 2017 to more than \$210 billion in 2020, at a CAGR of 11.9%*". Big Data does not only mean huge amount of data, but points to scenarios where data are diverse, come at high rates and must be proven to be trustworthy, as clarified by the 5V storyline [2]. The paradigm shift from traditional data mining approaches to Big Data techniques has already been acknowledged by the research community, which has introduced the concept of Big Data often provided as a service. Many works on Big Data techniques have focused on distilling the competences of Big Data technologists offering Platform-as-a-service solutions that support the automatic deployment of Big Data services [3], while others are focusing on supporting semi-automatic execution of analytics [4].

Today, however, we are nowhere near to Big Data commoditization. The approach to Big Data application development is usually bottom-up and the latest Big Data technology release drives the entire application development. This impairs the capacity of providing a solution that is technology independent, and only later on compiled on a specific technology (*technology neutrality*). Big Data toolkits do not provide a transparent design environment leading many developers to use them as black-boxes, with little or no insight on how analytics are actually executed. This makes the verification of the correct behavior of an analytics and its adherence to user's requirements a difficult and error-prone task (*verifiability*). Often, the execution workflow of a Big Data application is hidden to Big Data developers and architects, as well to final users. This impairs the capacity of providing accountable and reproducible solutions (*accountability and reproducibility*), reducing the performance of Big Data campaigns. Finally, the lack of an approach to Big Data commoditization substantially reduces the *usability and productivity* of Big Data processes, which are complex and difficult to manage.

In this paper, we try to fill in the above gaps. To this aim, we advocate a top-down approach where users' requirements and developers' models drive Big Data Analytics (BDA) development [5]. This approach must clearly go beyond the boundaries of traditional data modeling, which focused on resolving the complexity of relationships among data by means of schemata [6]. In fact, in addition to data representation, Big Data models should provide a shared specification of the process used to manage data resources, including data protection procedures and business rules, and of the computations to be done on them. They also need to provide all the information to carry out BDA over commodity execution platforms.

We propose a framework for Model-based Big Data Analytics-as-a-Service (MBDAaaS) [7], which, on one side, helps customers in deploying a full Big Data pipeline that addresses their requirements and, on the other side, provides full transparency on execution workflows and computations. We believe that there are clear advantages in adopting such a methodology in terms of shorting roll-

- *C.A. Ardagna, V. Bellandi, P. Ceravolo and E. Damiani are with the Dipartimento di Informatica, Università degli Studi di Milano, Milano, Italy. M. Bezzi and C. Hebert are with Security Research, SAP Labs France, Sophia Antipolis, France. Ernesto Damiani is also with CINI - Consorzio Interuniversitario, Nazionale per l'Informatica, Rome, Italy and EBTIC, Khalifa University, Abu Dhabi, UAE.*
  E-mail: claudio.ardagna,@unimi.it, valerio.bellandi@unimi.it, michele.bezzi@sap.com, paolo. ceravolo@unimi.it, ernesto.damiani@kustar.ac.ae, cedric.hebert@sap.com

out time and improving reuse. Our approach is based on a *declarative model* (Section 3), specifying the requirements of a given analytics in the form of goals describing the aim of the analytics and/or features to be supported by the execution environment. The customers navigate, select, and prioritize requirements through an interface, which guides them in selecting their preferences. Declarative requirements are then used to select a platform-independent *procedural model* (Section 4) specifying how analytics should be carried out and composed in terms of abstract services. Procedural models are finally compiled in a ready-to-be-executed *deployment model* (Section 5) specifying Big Data platform-dependent configurations and supporting automatic provisioning of computational components and resources. Such transformations are implemented in a methodology specifying a semi-automatic process for MBDAaaS (Sections 4 and 5). We practically evaluated our methodology in a real-world scenario that considers the threat detection system of SAP (Section 6), one of the largest Software and Information Technology services groups worldwide.

The contribution of this paper is threefold. First, we define a declarative approach to the specification of BDA requirements. Our approach includes a way to browse multiple options and express requirements as a set of computational-independent specifications. It also manages conflicts and interferences between specifications, providing a semi-automatic approach to their resolution. Second, we map our declarative models to a catalog of Big Data services, modeled in OWL-S [8], which represents the knowledge base over which the Big Data campaign is built. Relevant services are selected on the basis of the declarative specifications to produce a service composition driving the final deployment of the BDA on the target platform. Third, we implement a compiler that transforms the above platform-independent service composition in a platform-dependent workflow. The workflow is a ready-to-be-executed artifact that can be run on the target platform.

2 MBDAaaS Methodology

The methodology proposed in this paper aims to provide a MBDAaaS approach that decouples high-level specifications of a Big Data campaign from low-level details of the Big Data architecture. It is based on the following artifacts.

Declarative Specifications. They allow customers to shape a BDA in terms of computational-independent requirements and retrieve a set of services compatible with these models.

Service Catalog. It stores the set of abstract services (e.g., algorithms, mechanisms, components) that are available to Big Data customers and consultants for building their BDA. Each service defines an interface and a link to declarative specifications.

Service Composition Repository. It stores the service composition templates defining how abstract services can be composed to carry out the Big Data analytics. It supports traditional composition patterns, such as sequence, alternative, parallel.

Deployment Configurations. They permit to define the platform-dependent version of a service composition, as a workflow that is ready to be executed on the target Big Data platform.

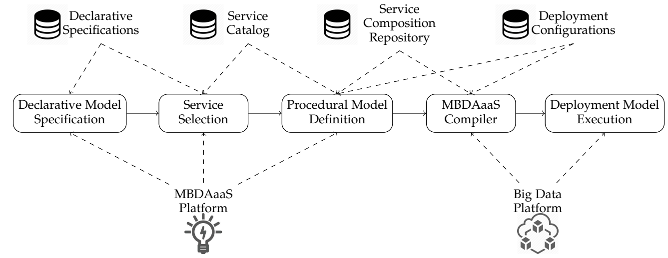


Fig. 1. MBDAaaS Methodology: Execution steps

A key aspect of our MBDAaaS methodology is the provisioning of a semantic-aware representation of Big Data service compositions.

Figure 1 presents the 5-step process supported by our methodology. In the first step (*Declarative Model Definition*), a Big Data customer defines a declarative model as a set of *Declarative Specifications* shaping a Big Data campaign (Section 3). To this aim, she uses a taxonomy of requirements that can be included in our declarative model.

In the second step (*Service selection*), after proving that declarative specifications are consistent, that is, they are not leading to mutually exclusive or interfering solutions (Section 3.2), concrete services compatible with the declarative model specified by the user are retrieved from the *Service Catalog* (Section 3.1).

In the third step (*Procedural Model Definition*), a Big Data consultant defines a platform-independent *Service Composition*, called procedural model, that composes (a subset of) compatible services selected at the previous step to form a Big Data campaign. *Procedural models* exploit the expressive power of OWL-S [8] to make an explicit representation of service compositions (Section 4.1). This supports query capabilities for retrieving services based on *interface type* [9], as it is possible to define parameters, return types, and data types using abstract classes (e.g., retrieve all services that return an anonymized data_set). In addition, verification procedures on *interface compatibility* [10] can be activated, as the parameters of the interconnected services have to match, and mismatches have to be reported to guide subsequent integration. For example, if a *data cleaning* service returning a *data_set* is interfaced with a *k-mean service*, requiring parameters *data_set*, k_m, *iterations*, a verification procedure has to prompt that k_m and *iterations* must be provided to proceed with the execution.

In the fourth step (*MBDAaaS Compiler*), MBDAaaS compiler uses the *Deployment Configurations* to transform the procedural model in a platform-dependent workflow called deployment model. This step is crucial to build a semi-automatic MBDAaaS and puts some strong constraints on the generality of the compiler, which needs to adapt to and configure the target Big Data platform.

Finally, in the fifth step (*Deployment Model Execution*), the target platform executes the designed analytics.

All data sets, models, and specifications used to develop our solution are available for interested readers at https://tinyurl.com/ycvh9u39. For the sake of readability, in the rest of this paper, we illustrate our work using code excerpts that highlight relevant aspects of our methodology.

3 Declarative Model Definition

The first step of our MBDAaaS methodology defines a *declarative model*. Declarative specifications include requirements that must undergo a consistency check to ensure a sound definition of a BDA.

3.1 Declarative Models

Declarative models are computational- and vendor-independent models allowing a customer to define a set of requirements shaping a BDA. *Requirements* define goals to be achieved by the BDA or features to be supported by the environment where the BDA will be executed. A *Goal* specifies an *Indicator*, which represents a way of measuring or assessing the goal, and an *Objective*, which represents the target to be achieved to consider the goal fulfilled. For example, a user can specify a goal about the *Analytics Aim* selecting the indicator *Task* and the objective *Regression*. A *Feature* identifies a functionality to be supported and may include *Sub-features* that can be mandatory/optional and inclusively disjoined/exclusively disjoined. For example, one can specify that a feature characterizing the data set is the *Data Model* that is further specified by a feature stating it as *Document Oriented*. Based on the systematisation proposed by several authors [11], [12], [13], we organized our declarative model in five main areas.

- **Data preparation** specifies all activities aimed to prepare data for analytics. For instance, it describes how to perform dimensionality reduction, or it defines how to guarantee data owner privacy using anonymization (e.g., hashing, k-anonymity).
- **Data representation** specifies how data are represented and expresses representation choices for each analysis process. For instance, it defines the data model (e.g., document-oriented, graph-based, key value) and partitioning (e.g., clustering, sharding).
- **Data analytics** specifies the analytics to be computed. For instance, it defines the expected outcome (e.g., descriptive, prescriptive, predictive) and the learning approach (e.g., supervised, unsupervised, semi-supervised).
- **Data processing** specifies how data are routed and parallelized. For instance, it defines the processing type (e.g., real-time, near real-time, batch) and the expected latency (e.g., low, medium, high).
- **Data visualisation and reporting** specifies an abstract representation of how the results of analytics are organised for display and reporting. For instance, it defines data display type (e.g., composition, order).

Requirements are specified according to a common vocabulary of goals and features available for interested readers at https://tinyurl.com/ycvh9u39, which maps the five conceptual areas above. It is represented in JSON-LD [14] format, to guarantee common semantics, reasoning capabilities, and interoperability among the different steps of our methodology.[1] Requirements can be prioritized and further refined by constraints. *Priorities* give an order between requirements, to resolve conflicts and inconsistencies at different stages of our methodology. *Constraints* support users with different competences in the specification of low-level settings, which will be applied to the parameters of the services in execution. A constraint is defined as an object data structure, that is, a collection of attributes including other objects or expressions of the form *op*(*attr*,*value*), where *op* is an operator in $\{=,\neq,<,>,\leq,\geq,\in,\notin\}$, *attr* represents an attribute referring to a procedural/deployment model, and *value* is a value (or an array of values) for the given attribute. A declarative model can then be defined as follows.

Definition 3.1 (Declarative model)**.** *A declarative model d consists of five elements (a_i, Φ), one for each conceptual area $a_i \in \mathcal{A}$, where Φ is a set of specification ϕ_i={r_i, $\mathcal{C}_i$, pr_i}, with r_i a requirement, $\mathcal{C}_i$={$c_1,\ldots,c_n$} a set of constraints on r_i driving the configuration of procedural and deployment models, and pr_i the priority of ϕ_i.*

Specifications ϕ_i can be defined at two levels:

- **requirement-level**, where ϕ_i={r_i,$-$,pr_i}
- **constraint-level**, where ϕ_i={r_i,$\mathcal{C}_i$,pr_i}.

Requirements model properties that are computational and platform independent, while constraints model properties that can be evaluated by executing a service within a specific platform. Requirements r_i can be naturally organized in hierarchies according to their two main types: *i)* $Goals$ ($\mathcal{G}$) that can be further specified by $Indicators$ ($\mathcal{I}$) and $Objectives$ ($\mathcal{O}$), denoted as ($\mathcal{G}$, $\mathcal{I}$, $\mathcal{O}$); *ii)* $Features$ ($\mathcal{F}$) that can be further specified by sub features, denoted as ($\mathcal{F}$, $\ldots$, $\mathcal{F}$). This implies that requirements r_i, and corresponding specification ϕ_i, can be defined at different levels of abstraction based on the hierarchy depth. We limit the hierarchy depth to three levels as follows:

- **first-level requirement**, where r_i=$\mathcal{G}$ or r_i=$\mathcal{F}$;
- **second-level requirement**, where r_i=($\mathcal{G}$,$\mathcal{I}$) or r_i=($\mathcal{F}$,$\mathcal{F}$);
- **third-level requirement**, where r_i=($\mathcal{G}$,$\mathcal{I}$,$\mathcal{O}$) or r_i=($\mathcal{F}$,$\mathcal{F}$,$\mathcal{F}$).

For instance, ϕ_i={(**Anonymization**, `Anonymization_Technique`, *k-anonymity*), {k_a>10}, 1} includes a third-level requirement restricting the required anonymization technique to k-anonymity, setting a constraint on the cardinality of parameter k_a of k-anonymity for r_i (i.e., k_a>10), and assigning to it the max priority (i.e., pr_i=1).

Example 3.1. *Figure 2 proposes an excerpt of a declarative model for area Data Preparation and Data Analytics. Goal* Anonymization *requires the input data set to undergo an anonymization process driven by goals anonymization model and anonymization techniques. It requires the adoption of a hashing techniques with constraints on the type of algorithm (i.e., SHA-256) and the target data field to be anonymized (i.e., UserID). Goal* Analytics Aim *requires a task implemented using*

1. A web application allowing the user to specify declarative requirements can be accessed at http://alphaplus.dti.unimi.it/mytoreador using credentials available at https://tinyurl.com/ycvh9u39.

```
{
  "@id": "http://www.toreador−project.eu/TDM/ps/mycustomid",
   "@context": {
      "s": "http://schema.org/",
      "tdm": "http://www.toreador−project.eu/TDM/"
   },
   [...]
      "tdm:preparation": {
      "@id": "http://www.toreador−project.eu/TDM/ps/areaPreparation",
      "@type": "tdm:Area",
      "tdm:label": "Data Preparation",
      "tdm:incorporates": [
    {
          "@type": "tdm:Goal",
          "tdm:label": "Anonymization",
          "tdm:incorporates": [
             {
                "@type": "tdm:Indicator",
                "tdm:label": "Anonymization Techniques",
                "tdm:incorporates": [
                   {
                      "@type": "tdm:Objective",
                      "tdm:constraint":
                      "{\"Algorithm\": \"SHA−256\",\"Target\": \"UserID\"}",
                      "tdm:label": "Hashing"
                   }
                   ] }
             ] } ] },
   [...]
   "tdm:analytics": {
      "@id": "http://www.toreador−project.eu/TDM/ps/area−analytics",
      "@type": "tdm:Area",
      "tdm:label": "Data Analytics",
      "tdm:incorporates": [
         {
          "@type": "tdm:Goal",
          "tdm:label": "Analytics Aim",
          "tdm:incorporates": [
             {
                "@type": "tdm:Indicator",
                "tdm:label": "Task",
                "tdm:incorporates": [
                   {
                      "@type": "tdm:Objective",
                      "tdm:label": "Crisp Clustering"
                   }
                ] }
             ] } ] },
}
```

Fig. 2. A fragment of the declarative model in JSON-LD

Crisp Clustering, that is, each clustered data point must belong to a single cluster.

3.2 Consistent Declarative Models

One of the key aims of MBDAaaS is guiding the user in the definition of consistent specifications along different Big Data conceptual areas. We then enrich declarative models with *Interference Declarations*, associating goals and features that are incompatible or interfere with each other.

3.2.1 Interference Declarations

We introduce a formal notion of *interference* as a way to detect inconsistent specifications and drive the declarative model verification process. Specifically, we define an interference as a relationship $in\{\phi_i, \phi_j\}\in\mathcal{IN}$, where ϕ_i and ϕ_j are two specifications, possibly, at different abstraction levels. We remark that different abstraction levels result in different enforcement points. Interferences at requirement level are enforced and solved at declarative level, while interferences at constraint level can be enforced either at declarative level, when constraints are defined as part of a declarative model, or at procedural level, when constraints are defined in the procedural model (see Definition 4.1). Moreover, enforcement can be executed either *i) prior to specification* (a priori) by automatically restricting the options available to a user based on previous selections, or *ii) posterior to specification* (a posteriori) by highlighting selections that are incompatible. A priori enforcement is preferable to increase system automation; a posteriori enforcement is preferable to increase user involvement and awareness. Another orthogonal dimension distinguishes among interferences that are enforced *i) deductively*, when the specifications included in the declarative models have a general application or *ii) inductively*, when specifications can be evaluated only by referring to the specific data set they apply to.

For example, to verify that a specification ϕ_i={(**Analytics_Aim**, `Model`, *Predictive*),−,−} is incompatible with ϕ_j={(**Visualisation_Operativity**, `Goal`, *Composition*),−,−}, we can simply refer to the requirements included in the declarative model. Instead, to verify that a specification ϕ_i={(**Analytics_Quality**, `True_Positive_Rate`, *medium*),−,−} is incompatible with ϕ_j={(**Analytics Aim**, `Task`, *Crisp_clustering*), {k_m: 5},−}, with k_m the number of clusters, we need to assess our data or at least a sample of the full data set.

3.2.2 Interference Enforcement

The interference enforcement process is modeled as a function that takes as input an interference $in\in\mathcal{IN}$ and produces as output a rule $rl\in\mathcal{R}$ that, applied to the specification with lower priority, resolves the interference.

Definition 3.2 (Interference enforcement process)**.** *Interference enforcement process is a function ξ:$\mathcal{IN}\rightarrow\mathcal{R}$ that takes as input an interference $in(\phi_i,\phi_j)\in\mathcal{IN}$ and returns as output the rule $rl\in\mathcal{R}$ to be performed on ϕ_j to produce a consistent declarative model. Interference enforcement can then be expressed as $\xi(\phi_i,\phi_j)=rl(\phi_i,\phi_j)$, where ϕ_j is the specification with lower priority and the inconsistency between ϕ_i and ϕ_j is resolved by restricting the domain of ϕ_j according to rl.*

Different rules can be defined to enforce interferences. A first common rule can be expressed in terms of boolean algebra, using the implication of a negation $P\rightarrow\neg Q$, which is more plainly translated in $\neg P\vee\neg Q$. This rule models simple scenarios where the specification of P forbids the specification of Q.

The boolean approach is however simplistic in many cases. To state a specification, we are required to express the intensity to which an indicator, a feature, or a constraint is required. In other words, we define a specific value in the domain of the values expressing the possible configurations of a specification ϕ. The specifications are then interpreted as variables and the rule rl is a function rl:(ϕ_i,ϕ_j)$\rightarrow\mathcal{D}(\phi_j)$ binding the domain of the requirement or the constraint in ϕ_i with the one in ϕ_j, denoted as $\mathcal{D}(\phi_i)$ and $\mathcal{D}(\phi_j)$, respectively. In other words, $\mathcal{D}(\phi_j)$ of ϕ_j is restricted according to $\mathcal{D}(\phi_i)$ of ϕ_i. Interpreting a rule as a function between two variables imply that if $x\in\phi_i$ is a value selected by the user, all the values $y\in\phi'_j$ with $rl^{-1}(y)>x$ cannot be proposed as an option to the user.

Our approach adopts a fuzzy definition of interference rules, defining a fuzzy interpretation of the logical connec-

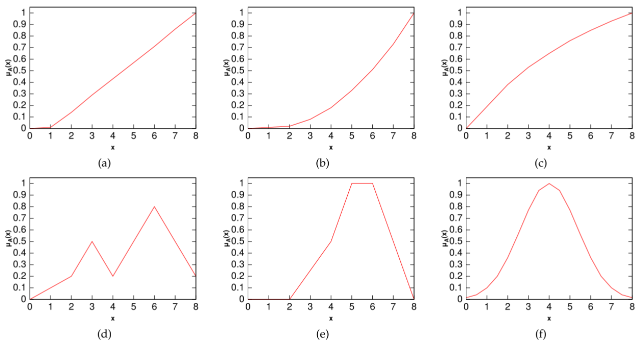


Fig. 3. Membership Functions that can be adopted to model interferences: a) linear monotonic; b) quadratic monotonic; c) squared root monotonic; d) triangular non-monotonic; e) trapezoidal non-monotonic; f) gaussian non-monotonic

tors of the rule and calibrating its behavior by normalizing the domains of the lower-level requirement or constraint in ϕ using a fuzzy predicate. The values of each ϕ can be mapped in $[0, 1]$, which represents the intensity of realization of ϕ. We are thus defining a membership function of a fuzzy predicate. The adoption of a step for interpreting the domain $\mathcal{D}(\phi)$ in terms of a fuzzy predicate, using a membership function, permits to decouple the enforcement of interferences from the definition of ϕ. The enforcement procedure is based on a single rule that interprets an interference as a logical implication of a negated. The behavior of the interferences depends on the membership function selected for fuzzifying requirement/constraint domain.

Figure 3 shows the different membership functions that can be adopted for interpreting a specification ϕ: linear monotonic (Figure 3(a)); quadratic monotonic (Figure 3(b)); squared root monotonic (Figure 3(c)); triangular non-monotonic (Figure 3(d)); trapezoidal non-monotonic (Figure 3(e)); gaussian non-monotonic (Figure 3(f)). All these functions express direct relationships; it is easy to see that inverse relationships can be represented as well.

Example 3.2. *We want to model an interference at constraint-level such as $in_a(\phi_1, \phi_2)$=({(***Accuracy***,* `False_Positive_Rate`*, medium), {f_p=v_1}, −}, {(***Anonymization***,* `Anonymization_Technique`*, k-anonymity, {k_a=v_2}, −)}). The rule rl is the fuzzy interpretation of $\neg P \vee \neg Q$. As illustrated in [15], different methods for interpreting logical operators in the fuzzy theory drive different composition methods. In this work we adopt a standard approach by interpreting the negation as the complement: $\neg x = 1 - x$; and the disjunction as the maximum: $x \vee y = \max(x, y)$. We have now to consider the domain of the possible configurations of ϕ_1 and ϕ_2 expressed as the interval of values (either discrete or continue) contained in a lower limit a and an upper limit b, $D : [a, b]$. Let us assume the domain of parameter f_p is in the range $[1, 6]$, the domain of parameter k_a is in the range $[1, 5]$, and v_1 and v_2 are the values selected by the user for the two parameters and triggering interference enforcement. We use a linear and monotonic membership function $\mu_A(v{\in}D){=}\frac{a-v}{b-a}$ to bind these values in the interval $[0, 1]$. This implies that if a user choose f_p=3 and k_a=3, their values are respectively equivalent to 0.4 and 0.5 in the domain $[0, 1]$. Because we know that in_a applies to the first term f_p, we can also infer important information on k_a, the second term of in_a. If we adopt the* a priori *approach, from the value v_1 specified for f_p, we can compute the set of values v_2 not allowed in k_a by taking the complement of $\mu_A(v_1)$, as we have $P \rightarrow \neg Q, P :: \neg Q$. This also implies that the complement of the first term is also a maximum boundary for the second term, that is, $\mu_A(v_2) \not\geq 1 - \mu_A(v_1)$. Following our example we have $1 - 0.4 = 0.6$ and then we know that all values $k_a{\geq}0.6$ cannot be chosen. If we adopt the* a posteriori *approach, we consider the values specified for f_p and k_a, and verify their consistency by taking the complement of $\mu_A(v_1)$, the complement of $\mu_A(v_2)$, and connect them using operator $\vee$, as $P \rightarrow \neg Q :: \neg P \vee \neg Q$. Assuming f_p=0.4 and k_a=0.5, we have $1{-}0.4{=}0.6$ and $1{-}0.5{=}0.5$, $max(0.6, 0.5){=}0.6$, that is, the max value that can be achieved by the membership value of the second argument of the interference. Since $k_a{<}1{-}f_p$ (i.e., $0.5{<}0.6$), the proposition is true and we can accept the value of k_a.*

4 Procedural Definition of a Big Data Campaign

The second step of our MBDAaaS methodology defines a *procedural model*. First abstract services in the Services Catalog, compatible with the specifications provided in the declarative model, are selected. These services (or a subset thereof) are then composed to provide a full description of the procedure to be executed by the platform.

4.1 Procedural Models

Procedural models are *platform-independent models* that formally and unambiguously describe how services must be configured and composed. They provide an arbitrary composition of services falling in the areas presented in Section 3.1, as formally defined below.

Definition 4.1 (Procedural Model)**.** *A procedural model m is a direct acyclic graph $G(V,E,\lambda)$, where a vertex $v_i{\in}V$ refers to a service in a specific area, an edge $(v_i,v_j){\in}E$ is annotated with function call f_i to the service represented by v_j, and $\lambda{:}V{\rightarrow}\mathcal{C}$ is a labeling function that associates a set $\{c_1,\ldots,c_n\}{\in}\mathcal{C}$ of constraints with each $v_i{\in}V$.*

We remark that constraints are used to set up service parameters with configuration values. For instance, c_i can specify the cardinality k_a of a data anonymization based on k-anonymity. We also remark that each function call annotating an edge in G triggers the execution of the corresponding mechanism in the deployment model, as described in Section 5.1.

4.2 OWL-S Service Composition

We use OWL-S [8] as the representation format for defining procedural models. OWL-S is a standard ontology providing a semantics to automatically discover, invoke, compose, and monitor services. It is structured in three interrelated sub-ontologies, known as *Profile*, *Process Model*, and *Grounding*. *Profile* expresses what the service “does”; *Process Model* describes “how it works”; and *Grounding* maps the constructs of the process model onto detailed specifications of message formats, protocols, and so forth. The definition of an OWL-S service composition, which is driven by the declarative model in Section 3, is built on the OWL-S ontology. It is a three-step process as follows.

OWL-S Service Definition. The definition of an OWL-S service composition includes the abstract description of OWL-S services that are compatible with the choices made in the declarative model. This phase is executed according to the following procedure. First, abstract services are defined using the OWL-S ontology. Each service includes a reference to a WSDL file that specifies the execution path of the service used at deployment time, that is, it identifies the concrete service to be executed. Second, each OWL-S service is manually connected to specifications in the declarative model using an extension to the OWL-S profile ontology and points to generic taxonomies or classification systems. Our extension provides two new elements `profile:serviceCategory` and `profile:categoryName`, where `profile:categoryName` defines a single specification and `profile:serviceCategory` is a container of elements `profile:categoryName`. These elements permit to specify a Boolean formula of specifications in a Disjunctive Normal Form (DNF), where different elements `profile:serviceCategory` are ORed, while different elements `profile:categoryName` in a single `profile:serviceCategory` are ANDed. Figure 4 proposes an excerpt where the OWL-S profile of an anonymization service is mapped to declarative specification *Data_Preparation.Anonymization_Technique.Hashing*, using a single element `profile:categoryName`.

OWL-S Service Selection. Upon receiving a declarative model, our methodology sequentially analyzes all OWL-S services and retrieves those services linked to Boolean formulas (of specifications) that are compatible with the requirements and constraints specified in the declarative model (step 2 in Figure 1). This is achieved using the extended OWL-S service description. For example, the anonymization service with OWL-S profile in Figure 4 is selected when the goal (**Anonymization**, `Anonymization_Technique`, *Hashing*) is specified in the declarative model for area data preparation. It is important to note that declarative models and their specifications are partial and incomplete by nature. The users can specify only a subset of available declarative requirements; also, the users only define the service goals of the Big Data campaign without specifying the execution flow. This scenario results in the selection of a set of compatible OWL-S services that: *i)* is a super-set of the services used in the Big Data campaign, *ii)* does not carry any information about how such services must be orchestrated.

OWL-S Service Composition. After retrieving the set of services compatible with declarative specifications, our process defines the OWL-S service composition (step 3 in Figure 1). Composite services are built by composing (a subset of) compatible services according to different operators such as `sequence`, `alternative`, `parallel` (see Section 5.2), using the control constructs provided by OWL-S. Given that, as discussed above, the declarative specifications are partial and incomplete by nature, a manual user intervention is required to resolve the inherent ambiguity due to the selection of a super-set of services and to the lack of information on the execution workflow. The user (or a consultant) is then responsible for manually drawing the OWL-S service composition. This is the only step where user intervention is required beyond declarative model definition.

```
<profile:serviceCategory>
    <profile:categoryName>
    Data_Preparation.Anonymization_Technique.Hashing
    </profile:categoryName>

    <profile:taxonomy>
      http://www.example.com/toreador/BDMOntologies/TDM.json
    </profile:taxonomy>
</profile:serviceCategory>
```

Fig. 4. Mapping of an OWL-S service for anonymization to a declarative specification

We note that OWL-S service composition carries all information about the process, data management, and orchestration, bringing back control to the final users and increasing the transparency of a Big Data computation.

Example 4.1. *We present a concrete example of our process for OWL-S service composition definition, starting from the definition of an OWL-S abstract service and its mapping to declarative model specifications. We consider the OWL-S modeling of the SHA-256 anonymization service of Spark, which is mapped to declarative requirements/-constraints according to the following Boolean formula in DNF:* [(**Anonymization**, `Anonymization_Model`, differential_privacy) $\wedge$ Algorithm:SHA-256] $\vee$ [(**Anonymization**, `Anonymization_Technique`, hashing) $\wedge$ Algorithm:SHA-256]. *Again, an excerpt of this mapping is provided in Figure 4.*

*Figure 5 presents the OWL-S process model of the anonymization service, where a data set (*Dataset*) and the parameter(s) target of the anonymization algorithm (*field*) are taken as input (elements* `process:hasInput`*), and the anonymized data set (*AnonymizedDataset*) is returned as output (element* `process:hasOutput`*).*

*Figure 6 presents an example of grounding, where the function of the anonymization process is specified. In our example, a reference to the WSDL file of the service (*HashingAnonymizationService.wsdl*) is used as a placeholder that points to the functionality/library provided by the target Big Data platform and implementing the service itself. This reference is fundamental to map the abstract OWL-S service to the real service in the target platform, linking a central repository of information needed to execute the real service (see Section 5).*

Finally, Figure 7 presents a basic service composition that composes in a sequence a data cleaning service, the anonymization service we just introduced, and a clustering service aimed to segment data based on a distance metric.

5 Executable Workflows

The last step of our MBDAaaS methodology is based on a compiler that automatically transforms a procedural model (OWL-S service composition) in a deployment model (executable workflow) that is ready to be executed on the target Big Data platform. We note that OWL-S service composition is a technology-independent representation of

```
<process:AtomicProcess rdf:about="#AnonymizationService:Process">
      <process:hasOutput>
         <process:Output rdf:ID="Dataset">
            <process:parameterType rdf:datatype="http://www.w3.org/2001/
XMLSchema#anyURI">
                 &tdm;#AnonimyzedDataset
            </process:parameterType>
         </process:Output>
      </process:hasOutput>
      <service:describes rdf:resource="#AnonymizationService"/>
      <process:hasInput>
         <process:Input rdf:ID="Dataset">
            <process:parameterType rdf:datatype="http://www.w3.org/2001/
XMLSchema#anyURI">
                 &tdm;#Dataset
            </process:parameterType>
         </process:Input>
      </process:hasInput>
      <process:hasInput>
         <process:Input rdf:ID="Field">
            <process:parameterType rdf:datatype="http://www.w3.org/2001/
XMLSchema#anyURI">
                 &tdm;#Field
            </process:parameterType>
         </process:Input>
      </process:hasInput>
      <rdfs:label>Process</rdfs:label>
   </process:AtomicProcess>
```

Fig. 5. Anonymization Service: OWL-S process model

```
<grounding:WsdlGrounding rdf:about="#Grounding">
   <service:supportedBy rdf:resource="#Service"/>
   <grounding:hasAtomicProcessGrounding>
      <grounding:WsdlAtomicProcessGrounding
      rdf:ID="AtomicProcessGrounding"/>
   </grounding:hasAtomicProcessGrounding>
</grounding:WsdlGrounding>
<grounding:WsdlAtomicProcessGrounding
rdf:about="#AtomicProcessGrounding">
   <grounding:wsdlInput>
      <grounding:WsdlInputMessageMap>
         <grounding:owlsParameter rdf:resource="#Dataset"/>
         <grounding:wsdlMessagePart rdf:datatype="http://...#anyURI">
            file:HashingAnonymizationService.wsdl#Dataset
         </grounding:wsdlMessagePart>
        </grounding:WsdlInputMessageMap>
    </grounding:wsdlInput>
<grounding:wsdlOutput>
   <grounding:WsdlOutputMessageMap>
      <grounding:owlsParameter
      rdf:resource="#HashingAnonymizedDataset"/>
      <grounding:wsdlMessagePart rdf:datatype="http://...#anyURI">
         file:HashingAnonymizationService.wsdl#anonymizedDataset
      </grounding:wsdlMessagePart>
   </grounding:WsdlOutputMessageMap>
</grounding:wsdlOutput>
```

Fig. 6. Anonymization Service: OWL-S grounding.

the executable workflow, which models what the Big Data campaign should achieve and how to achieve its objectives; the executable workflow is the implementation of an OWL-S service composition on a specific technology/platform.

5.1 Deployment Models

Deployment models are executable, platform-dependent models that specify how procedural models are instantiated and configured on a target platform, using components relevant for the analytics to be performed. They define configurations for architecture deployment and drive analytics execution in real scenarios. A deployment model $\overline{G}$ is an instance of a procedural model G in Definition 4.1 and is generated according to an exogenous vertical transformation [16].

```
<process:CompositeProcess rdf:about="tdm:CompositeProcess00001">
  <process:composedOf>
    <process:Sequence>
      <process:components>
       <process:ControlConstructList>
        <list:first> <process:Perform>
          <process:process rdf:resource="tdm:DataCleaningService"/>
        </process:Perform> </list:first>
      <process:components>
       <process:ControlConstructList>
        <list:first> <process:Perform>
          <process:process rdf:resource="tdm:AnonymizationService"/>
        </process:Perform> </list:first>
      <process:ControlConstructList>
        <list:first> <process:Perform>
         <process:process rdf:resource="tdm:ClusteringService"/>
        </process:Perform> </list:first>
       </process:ControlConstructList>
      </process:components>
    </process:Sequence>
  </process:composedOf>
</process:CompositeProcess>
```

Fig. 7. OWL-S service composition

Definition 5.1 (Deployment model transformation $\bowtie$)**.** *Let $m \in \mathcal{M}$ be a procedural model and $\{c_1, \ldots, c_n\} \in \mathcal{C}$ a set of constraints on deployment models in Definition 3.1. Model transformation $\bowtie: \mathcal{M} \times \mathcal{C} \rightarrow \mathcal{M}_d$ is a function that takes the procedural model $m = G(V, E, \lambda)$ and the constraints $\{c_1, \ldots, c_n\}$ as input, and produces a deployment model $m_d = \overline{G}(\overline{V}, \overline{E}, \lambda)$ as output. Deployment model m_d is such that the following conditions hold:*

1) *m_d is a super-model of m (denoted $m \sqsubseteq_m m_d$), where $V \subseteq \overline{V}$, $E \subseteq \overline{E}$;*
2) *each vertex $\overline{v}_j \in \overline{V}$ of m_d instantiates and configures corresponding vertex $v_i \in V$ of m according to constraints $\{c_1, \ldots, c_n\}$; and*
3) *for each vertex $v_i \in V$, corresponding vertex $\overline{v}_j \in \overline{V}$ must be such that $\lambda(\overline{v}_j)$ is consistent with $\lambda(v_i)$.*

We remark that there is a clear isomorphism between procedural model m and deployment model m_d. We also note that each $\overline{v}_j \in \overline{V}$ is instantiated with a Big Data algorithm, service, or component. We finally note that, during the transformation, a single vertex $v_i \in V$ could be instantiated in multiple vertices $\overline{v}_j \in \overline{V}$. In this case, condition 1 checks the satisfaction of constraints $\{c_1, \ldots, c_n\}$ on a sequence of vertices $\overline{v}_j \in \overline{V}$; condition 2 checks consistency between $\lambda(v_i)$ and $\lambda(\overline{v}_j)$ for all $\overline{v}_j$. Applying to generic direct acyclic graphs, our transformation can be used in different scenarios. For instance, in case the analytics show a true data dependency, the procedural and deployment models can be represented as data flow models and the transformation maps each vertex $v_i \in V$ in one or more vertex $\overline{v}_j \in \overline{V}$. Otherwise, in case there is not a strict data dependency, the procedural and deployment models can be represented in terms of features and modeled as feature models, where the transformation extends leaf nodes in the procedural model with a subtree rooted at such nodes in the deployment model.

5.2 Compiler

The transformation in Definition 5.1 is implemented by a compiler (step 4 in Figure 1) that takes as input the OWL-S service composition returned in step 3, the OWL-S service

TABLE 1
Control constructs considered by the compiler

Operators	OWL-S Constructs	BPEL Constructs	Oozie Constructs
Sequence $\odot$	`⟨process:Sequence⟩` ... `⟨/process:Sequence⟩`	`⟨sequence ...⟩` ... `⟨/sequence⟩`	–
Alternative $\otimes$	`⟨process:Choice⟩` ... `⟨/process:Choice⟩`	`⟨pick ...⟩` ... `⟨/pick⟩` `⟨switch⟩` ... `⟨/switch⟩`	`⟨switch⟩` ... `⟨/switch⟩`
Parallel $\oplus$	`⟨process:Split⟩` ... `⟨/process:Split⟩`	`⟨flow ...⟩` ... `⟨/flow⟩`	`⟨fork name="forking"⟩` ... `⟨/fork⟩`
Loop μ	`⟨process:Iterate⟩` ... `⟨/process:Iterate⟩`	`⟨while ...⟩` ... `⟨/while⟩`	`⟨action name="loop"⟩` ... `⟨/action⟩`

description of all services selected in step 2 (Figure 7) and information on the target platform, and produces as output a technology-dependent workflow that can be executed by the workflow engine available on the target platform. The compiler represents the cornerstone of our MBDAaaS methodology. It consists of two main sub-processes, namely, *structure generation* and *service configuration*, each focusing on a specific part of the transformation as follows.

Structure generation. The compiler parses the OWL-S service composition and identifies the process operators composing it. Different operators can be managed including:

1) *Sequence* $\odot$ implementing a sequence statement where s_i is executed before s_j ($s_i \odot s_j$).
2) *Alternative* $\otimes$ implementing a conditional statement where either s_i or s_j is executed ($s_i \otimes s_j$).
3) *Parallel* $\oplus$ implementing a parallel statement where both s_i and s_j are executed at the same time ($s_i \oplus s_j$).
4) *Loop* μ implementing a loop statement, which can be either implemented as a separate statement (μs_i) or as a degeneration of the sequence.

Upon identifying the operators, the compiler loads the driver of the language used for deployment model specification and generates an empty skeleton of the Big Data pipeline to be executed, which corresponds to the structure of the procedural model. The structure, in addition to operators, contains an empty service for each service in the procedural model. It is important to remark that our approach has been designed to be generic, meaning that our compiler is independent of the selected workflow language. The compiler, in fact, can translate an OWL-S service composition in a workflow that is modeled both using a workflow language specifically defined for Big Data, such as Apache Oozie,[2] and a workflow language originally defined for web service composition, such as WS-BPEL. Table 1 presents the mapping between OWL-S operators and corresponding operators in Apache Oozie and WS-BPEL workflow languages. In the following, for simplicity but with no lack of generality, we focus on Oozie workflows.

Service configuration. After generating the workflow structure, for each empty service s'_i, *i)* the corresponding service s_i in the procedural model is identified, *ii)* service s_i is instantiated using the OWL-S service description, information on the Big Data execution platform, and the language selected for specifying the deployment model. We recall that the reference to a WSDL file in the OWL-S service description is used to identify the real service instance to be added in s'_i and to be executed on the target platform.

2. Other workflow languages available in the market are AirBnB Airflow (https://airflow.apache.org), LinkedIn Azkaban (https://azkaban.github.io), or Spring Cloud Data Flow (https://cloud.spring.io/spring-cloud-dataflow).

5.3 From OWL-S service compositions to Oozie workflows

Structure generation and service configuration in Section 5.2 are implemented using Oozie workflows.

5.3.1 *Structure generation*

We use a SAX parser to parse the OWL-S service composition by means of an event-driven online algorithm and generate the corresponding Oozie workflow structure. An OWL-S service composition, modeled using the process model sub-ontology and the operators in Table 1, defines `process:CompositeProcess` as the root of the composite process. It then defines the inputs of the process with element `process:hasInput` and the corresponding outputs with element `process:hasOutput`. It finally describes the real service composition using element `process:composedOf`. The latter element specifies how services are composed and corresponding inputs/outputs are linked among them. In particular, it includes the specific operator (e.g., element `process:sequence`), which in turn contains the specific component services (element `process:components`) to be composed. For instance, Figure 7 shows an example of sequence (element `process:ControlConstructList`), where three services are composed in a sequence on a pair basis. The first service, *DataCleaningServices*, is wrapped within an element `list:first` and composed with a sequence of services *AnonymizationService* and *ClusteringService* which are wrapped within element `list:rest`.

The result is an OWL-S document that gets translated into an Oozie workflow according to the control constructs in Table 1. First of all, element `workflow-app` is added as the root of the Oozie workflow. Then, for each service in the OWL-S service composition, an element `action` is added within the root. Each of the generated elements has an attribute *name*, and two elements `ok` and `error` modeling the execution flow. Figure 8 presents an Oozie workflow that corresponds to the OWL-S composition in Figure 7. In our example, three elements `action` have been added at the root level. The first action has name *DataCleaningService* and is connected to *AnonymizationService* using element `ok`, which, in turn, is connected to *ClusteringService* using element `ok`. Action *ClusteringService* is then connected to the end of the workflow. A default service (*kill_job*) is modeled using element `kill` and represents an unexpected end of the workflow due to faults/errors.

5.3.2 *Service configuration*

Service configuration acts on each element `action` and configures it according to the information included in the OWL-S sub-ontologies Profile and Grounding. The first element specifies, for each action, the language used to implement the service itself (e.g., `Java` or `Spark`). Then, different elements specify how to execute the job including: *i)* `job-tracker`, to trace and manage the job, *ii)* `name-node`, where the job is executed,

```
<workflow−app name='Process1' xmlns='uri:oozie:workflow:0.1'>
  <start to='DataCleaningProcess' />
  <action name='DataCleaningProcess'>
    <java>
      <job−tracker>${jobTracker}</job−tracker>
      <name−node>${nameNode}</name−node>
      <prepare><delete path='${outputDir}'/></prepare>
      <configuration>
        <property>
          <name>mapred.reduce.tasks</name>
          <value>300</value>
        </property>
      </configuration>
      <main−class>${MainClass}</main−class>
      <arg>${normalization}</arg>
      <arg>${datasetURI}</arg>
    </java>
    <ok to='AnonymizationProcess' />
    <error to='kill' />
  </action>
  <action name='AnonymizationProcess'>
    <java>
      <job−tracker>${jobTracker}</job−tracker>
      <name−node>${nameNode}</name−node>
      <prepare><delete path='${outputDir}'/></prepare>
      <configuration>
        <property>
          <name>mapred.reduce.tasks</name>
          <value>300</value>
        </property>
      </configuration>
      <main−class>${MainClass}</main−class>
      <arg>${normalizedDatasetURI}</arg>
      <arg>${hashing}</arg>
      <arg>${IPAddress}</arg>
    </java>
    <ok to='ClusteringProcess' />
    <error to='kill' />
  </action>
  <action name='ClusteringProcess'>
    <java>
      <job−tracker>${jobTracker}</job−tracker>
      <name−node>${nameNode}</name−node>
      <prepare><delete path='${outputDir}'/></prepare>
      <configuration>
        <property>
          <name>mapred.reduce.tasks</name>
          <value>300</value>
        </property>
      </configuration>
      <main−class>${MainClass}</main−class>
      <arg>${anonDatasetURI}</arg>
      <arg>${K}</arg>
      <arg>${maxIterations}</arg>
    </java>
    <ok to='end' />
    <error to='kill' />
  </action>
  <kill name="kill">
    <message>Task failed, error message[${wf:errorMessage()}]</message>
  </kill>
  <end name='end'/>
</workflow−app>
```

Fig. 8. Example of an Oozie workflow

iii) `prepare`, to cleanup the directory for job execution, *iv)* `configuration`, the configurations of the job to be executed, *v)* `main − class`, the class to be executed, *vi)* a set of `arg`, arguments to be given as input to the job.

Example 5.1. *Figure 8 shows an example of an Oozie workflow corresponding to the OWL-S workflow in Figure 7 and compatible with the Oozie workflow engine. The workflow in Figure 8 is composed of three activities (elements* `action`*) in a sequence that execute* i) *data cleaning,* ii) *anonymization, and* iii) *clustering. As indicated by the element* `start`*, the workflow begins by executing* DataCleaningProcess*, which receives as input two parameters (elements* `arg`*), the first with the type of normalization to be executed (${normalization}) and the second with the target data set (${datasetURI}). Upon a successful execution of* DataCleaningProcess, AnonymizationProcess *is executed (second element* `action`*). It receives as input the output of* DataCleaningProcess *(${normalizedDatasetURI}), the anonymization operation (${hashing}) and the field(s) over which the anonymization must be performed (${IPAddress}). Upon a successful execution of* AnonymizationProcess, ClusteringProcess *is finally executed (third element* `action`*). It receives as input the results of* AnonymizationProcess *(${anonDataset}), and parameter(s) for clustering (${K}, ${maxIterations}).*

There is, however, a subtlety to consider: the workflow in Figure 8 is not completely specified. In fact, the activities in the workflow refers to templates that need to be instantiated with parameters in the OWL-S workflow.[3] *For instance, let us consider the reference to* AnonymizationProcess *in Figure 8. The corresponding activity is completely specified unless the variables associated with the parameters anonymization operation ($hashing) and target ($IPAddress). When $hashing and $IPAddress are filled in with real values, the workflow is completely specified and ready to be executed.*[4]

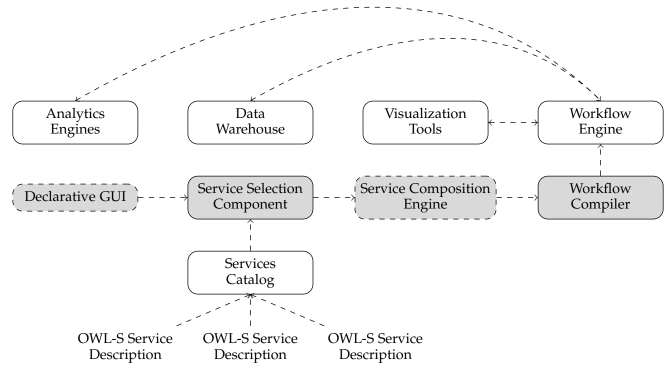


Fig. 9. Extended Big Data platform architecture implementing MBDAaaS in Figure 1

6 Experimental Evaluation

We present the application of our methodology in a real-world industrial scenario. To this aim, we integrated our methodology, presented in Figure 1, within traditional Big Data platforms, by changing the architectural flow between their components. Figure 9 shows the architectural extension to a "standard" Big Data platform, where gray boxes refer to components implementing our methodology; white boxes refer to traditional components of a Big Data platform used to execute the services selected in the first stage of our methodology; boxes with dashed lines refer to steps of the methodology that require a manual configuration.

According to Figure 9, the users first specify the declarative model using the web-app form provided by the *Declarative GUI*. The JSON-LD file (Figure 2) returned as output is

3. In case a parameter value is missing, it is prompted to the user that can provide it.

4. In case no workflow engine is available on the target platform, these parameters are directly added in the call to the corresponding library (through a command line interface).

given as input to the *Service Selection Component*, and used to configure traditional Big Data platform components: *Analytics Engines*, *Data Warehouse*, *Visualization Tools*. Upon receiving the declarative specifications, *Service Selection Component* automatically retrieves the compatible services, which are manually composed by the users to build an OWL-S service composition using the *Service Composition Engine*. The OWL-S service composition is given as input to the *Workflow Compiler*, which automatically transforms it into a workflow that can be managed by the selected *Workflow Engine*. The workflow is used to finalize the configuration of traditional Big Data components and is executed by the *Workflow Engine* on the selected platform. Retrieved results can be used to tune the declarative model specification and restart the process from scratch.

In our experimental evaluation, we adopted the Hadoop stack that is based on the YARN programming model and the Hadoop Distributed File System (HDFS). We then used Apache HBase as our Data Warehouse, a database engine built using Hadoop and modeled after Google's Big Table. We further integrated Apache Spark as our Analytics Engine, a general purpose, fast, and reliable cluster computing engine. We then used Zeppelin as the visualization tool for data sets and results. We finally adopted Oozie as our workflow engine and integrated it with the rest of the Hadoop stack.

All artifacts used to run this experiment, as well as the results obtained, are available at https://tinyurl.com/ycvh9u39.

6.1 Reference Scenario: Threat Detection Systems

Threat detection and prevention in software ecosystems [17] have evolved from the network level and start encompassing the application layer. Threat Detection Systems (TDS) detect potential attacks on the application landscape by gathering and analyzing log data, such as user change logs, security audit logs, remote function call gateway logs, and transaction logs. Logs are pre-processed, anonymized, translated into a common format, and analyzed by pattern or anomaly detection algorithms, which can highlight suspicious events. On top of the generated events and alerts, a detailed investigation is performed by a human expert to decide if a real attack was detected or was a false positive. However, with the increasing size and complexity of software systems, the volume and diversity of log data are becoming major issues. Customers have in place a large spectrum of different systems and a wide range of data security policies. As a result, including and managing these heterogeneous log files currently need a significant customization effort, especially when they contain sensitive and personal information (e.g., user IDs, IP addresses), come from logs of multiple customers, or are accessed via a third party (e.g., cloud provider) running the TDS. Similarly, customers often need different security analysis, depending on the security context, industrial sector, risk management policies, and the analytics functionality need to be often customized too. Accordingly, major challenges for data analysis by TDS systems are related to devising a flexible framework supporting: *i)* the provisioning of customized analytics and reporting approaches for stakeholders; *ii)* data anonymization (at different levels, for different customers, for different purposes); *iii)* a scaling approach for variable data loads; *iv)* a limited effort (i.e., semi-automatic) for the integration of new and diverse log files, which also adapts corresponding analytics.

We present how MBDAaaS approach can be used in the real in-production TDS of SAP, one of the largest IT enterprises worldwide, and its extension to include novel analytics. The major challenge of TDS, when searching for security incidents, lies in the ability to detect either a deviation from a standard behavior (unplanned anomalous activity), during or outside of an exceptional process (planned anomalous activity), or regular malicious activities merged into the normal state of operations (unplanned ordinary activities such as advanced persistent threats or repeated frauds).

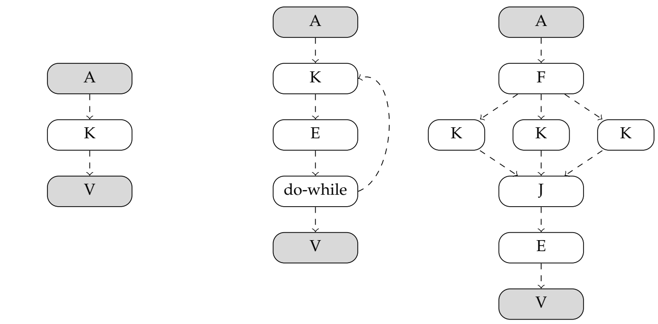
6.2 Experimental Evaluation

We extended the preliminary evaluation conducted in [18] by executing a complete process from declarative model specification to deployment model generation and execution. Our starting point was devising an anomaly detection functionality through clustering detected anomalies. In particular, we aimed to detect usage of certain types of programs (called transactions) outside of a user's peer group (possibly a case of privilege escalation) and to cluster anomalies pertaining to a planned abnormal activity in a single stage (instead of assessing them one by one, for example anomalies due to a system upgrade or due to an internal reorganization). We used a log data set containing around six weeks of activity collected from SAP systems deployed in a test environment. We considered three features, namely Executed Transaction, Data Sent, and Data Received, extracted from the log files and aggregated per user. For testing purpose, we artificially generated a data set with multi-variate Gaussian distribution having the same average and covariance matrix as the original data points.

In the following, we describe a step-by-step application of our methodology to this scenario.

Declarative model specification and interferences. The declarative model for our TDS scenario mainly defines three requirements: *i)* log data set must address data minimization privacy requirement by k-anonymizing the *UserID* field with k_a=5, while guaranteeing full analytics quality; *ii)* data must be analyzed using a clustering algorithm; *iii)* data visualization must provide scatter-plots, supporting visual inspection of data by expert users, to select valid alerts.

Interference rules are then checked to evaluate declarative model consistency. In our example, a rule is triggered between specification ϕ_1={(**Anonymization**, `Anonimization_Technique`, *k-anonymity*), {k_a:5, Target: UserID}, 0.8} and ϕ_2={(**Analytics_Quality**, `False_Positive_Rate`, *low*), {f_p: 1}, 1}. Since *Analytics Quality* has higher priority, the fuzzy rule in Figure 3(b) is applied and the domain of k_a of k-anonymity is restricted to k_a:1. In this case, since anonymity is not preserved anymore, the final user is prompted and a new specification for goal *Anonymization Technique* is provided: ϕ_1=(**Anonymization**, `Anonimization_Technique`, *hashing*), {Algorithm: SHA-256, Target: UserID}, $-$}.



(a) Single value for k_a (b) No value for k_a (c) Range value for k_a

Fig. 10. OWL-S service compositions using anonymization (A), k-means (K), elbow (E), and visualization (V) services. Fork (F), Join (J), and do-while represent operator nodes

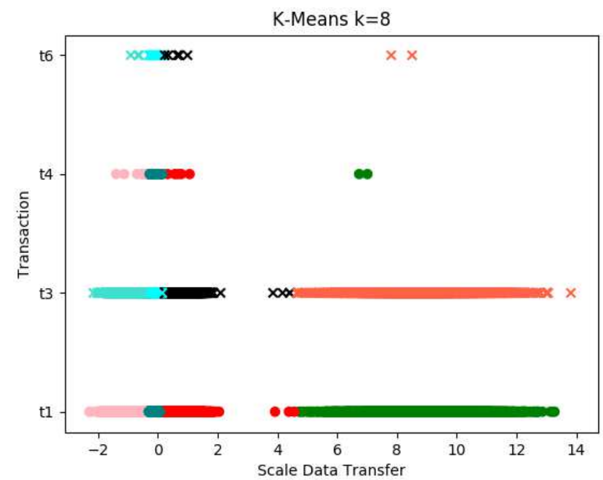


Fig. 11. Two-dimensional projections of analytics results

Service selection. Declarative model specification is used to select the set of compatible services that can be composed to build the Big Data campaign. According to the three requirements above, our MBDAaaS methodology selects all SHA-256 services, different instances of clustering services, and all visualization services, including the ones provided by Zeppelin, for scatter-plot visualization, available in the target platform. The specification of declarative constraints can further affect the list of selected services. In our example, the definition of constraint k_m, that is, the number of clusters, affects service selection results. When k_m is not defined or is defined as a range (Figure 2), the list of compatible services also includes optimization services such as *elbow method* and *silhouette score*. This means that either the final user can select a value for k_m when specifying the procedural model or an optimization algorithm can be used to infer it.

Service composition. Upon service selection, the final user manually composes a subset of the selected services to build her Big Data campaign. In our scenario, the user selects a SHA-256 anonymization service, a k-means clustering service, and a scatter-plot visualization service. There is, however, a subtlety to consider when services are composed: selections made at declarative level could also pose restrictions on the way in which services can be composed. Let us consider the case for the selection of constraint k_m (i.e., number of clusters) of k-means.

- *Single value.* When the value of k_m is a single value, no optimization services will be retrieved and composed. This case is supported by our service selection step (Figure 10(a)).
- *No value.* When k_m is not defined, optimization algorithms are retrieved in the service selection step. In this case, a *do-while* loop must be used to build our OWL-S services selection, where at each step a k-means execution is followed by an optimization step (e.g., elbow method) that also considers the result of the previous step. The loop ends according to the optimization function (Figure 10(b)).
- *Range.* When k_m is defined as a range, optimization algorithms are retrieved in the service selection step. Similarly to the previous case, a *do-while* loop can be used, though it could be not effective. In fact, knowing the different k_m, a parallel structure can be used to execute k_m times the k-means algorithm with incremental k_m, evaluating the optimization function at the join node of the parallel structure (Figure 10(c)).

Figure 10 shows the flows of the three possible OWL-S service compositions. We remark that service compositions are manually defined by the user according to the retrieved compatible services. In particular, in our experimental evaluation we selected the composition in Figure 10(b). This is because the declarative model provided as input (as an extension of the excerpt in Figure 2) does not specify the number of clusters k_m, meaning that the user does not know the optimal parameter and optimization services, such as *elbow method*, are beneficial.

Workflow execution. The composition in Figure 10(b) is finally given as input to our compiler that automatically generates the Oozie workflow. The workflow is then stored in our platform, following the appropriate path and instantiated according to the information available in the files $job.xml$ and $properties.xml$. These files are associated with each service component, so that the Oozie engine can refer to them when executing.

Initial analysis. Figure 11 presents the results retrieved through our MBDAaaS using the Big Data platform in this section. Our experiments returned 8 clusters, according to the optimum k_m returned by elbow method, representing similar events. The clustering was able to group all planned anomalous activities (peaks in data transfer, observed regularly for all users) into a single cluster (scale data transfer>4), filtering out a large number of events which would otherwise have triggered false-positive alerts.

Refinement. One of the strengths of the MBDAaaS lies in the ability to quickly redeploy alternative analytics. Leveraging this capability, we analyzed our data set with a Gaussian Mixture Model (GMM) instead of k-means. Since our GMM implementation does not require an elbow method for finding an optimal value for k_m, the resulting OWL-

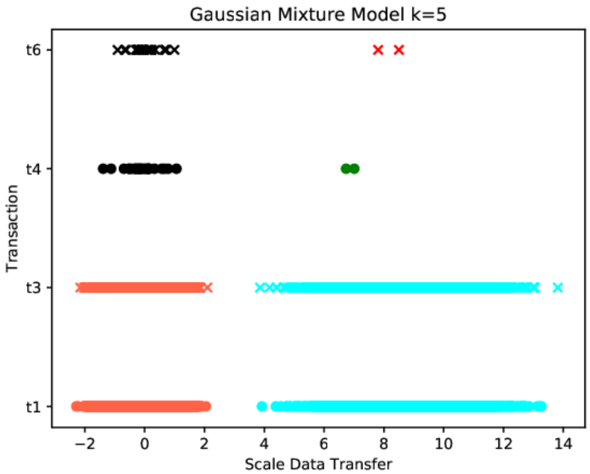


Fig. 12. Two-dimensional projections of analytics results after refinement

S composition was simplified as an Anonymization step, followed by a GMM step and by a Visualization step, akin to Figure 10(a). This second experiment returned 5 clusters presented in Figure 12. GMM clustering permitted to group regular transactions in a limited number of clusters and highlighted new small clusters, corresponding to a transaction seldom used by one of our users, which was not visible via k-means clustering. This cluster corresponds to a second type of anomaly: a transaction run by a user but by no other users having a similar role. We then considered the segmentation produced by GMM more significant.

Performance evaluation. We evaluated the overhead introduced by our solution with respect to standard Big Data architectures, that is, the overhead introduced by the service composition engine and the workflow compiler. The performance of the service composition engine and compiler have been measured in the three scenarios in Figure 10, using a laptop with a CPU Intel i7 and 8GB of RAM, retrieving the following results. The service composition engine generated the OWL-S of the composition in Figure 10(a) in 300ms, the composition in Figure 10(b) in 310ms, and the composition in Figure 10(c) in 380ms. Then, the workflow compiler generated the Oozie workflow of the composition in Figure 10(a) in 420ms, the composition in Figure 10(b) in 520ms, and the composition in Figure 10(c) in 780ms. These results show the marginal overhead introduced by our approach when developing a Big Data campaign.

6.3 Discussion

We presented a complete walkthrough of our MBDAaaS methodology, from declarative model specification to deployment model execution and refinement, to show its feasibility and usefulness. The entire process was triggered by the definition of the declarative model that permitted to carry out the expected Big Data campaign. In particular, we would like to stress the following points. First, the requirements specified by the user, though partial and incomplete (with only ~2% of the requirements specified in our experiments), can be sufficient to select a set of services that, appropriately composed, produce a significant result. The interference enforcement on declarative specifications reduces the number of scenarios where declarative specifications bring to Big Data campaigns producing wrong or useless results. The details about the process, data management, and orchestration of the specific procedural implementation permit to compare different alternative solutions before their deployment (see Figure 10), paving the way to the a priori assessment of the impact that alternative specifications have on performance or quality of analytics. Also, such details provide all information needed to define privacy-by-design Big Data campaigns, by verifying OWL-S service compositions against existing standards (e.g., General Data Protection Regulation – GDPR).

In conclusion, our methodology supports users in the management of Big Data campaigns that require tuning, adaptation, and continuous analytics. More specifically, It contributes to fill in the gaps identified in Section 1.

- **Usability and productivity**: it supports users lacking Big Data expertise in managing Big Data analytics deploying a full Big Data pipeline. It supports fast roll-out with fine-tuning of the declarative model, implementing an efficient link between R&D and production.
- **Accountability and reproducibility**: it supports the comparison of multiple solutions, with a clear specification of services. It supports the reuse of models across domains and technologies, or their refinement on the basis of collected results by applying MBDAaaS iteratively.
- **Verifiability**: It supports a straightforward integration with assurance techniques, assessing preconditions and checking consistency with requirements in the declarative model.
- **Technology neutrality**: It supports platform-independent design and development.

Our MBDAaaS methodology has some room for improvement mainly in terms of flexibility and validation capability, which we leave for our future work. First, while our proposal provides a highly-usable approach for users with low competences, it falls short to satisfy requirements of skilled users that need also to control the low-level parameters of the computation before moving to production. Second, our approach does not permit to select services at infrastructure layer hosting the computation on the basis of declarative specifications. Our approach should then include execution-side use of declarative model information for generating *deployment recipes* supporting service distribution across virtualized containers like Dockers. This perspective, fully compatible with our methodology, will introduce a degree of automation in execution. Finally, a throughout benchmarking of our methodology will be targeted to evaluate the quality of the retrieved results on the basis of the selected technologies, the level of details in the declarative model, and the type of data and analytics selected.

7 Related Work

Research on Big Data-as-a-Service (BDaaS) [11], [19], [20] has been tackled from four main perspectives, which vary on the degree of automation supporting the execution of a Big Data campaign.

Big Data Platform-as-a-Service considers those approaches offering support for deploying the core platform (e.g., Hadoop and Spark with interactive query services) as a service [21]. Azure and Amazon are probably the most prominent representatives of this perspective.

Performance-Oriented BDaaS considers approaches where the deployment of a Big Data campaign is guided by optimization criteria aimed to take the computational workload under control [22], [23].

Application-Oriented BDaaS considers those approaches where the deployment of a Big Data campaign is guided by specific business cases, including requirements on data management or analytics to be executed. Although this perspective shaped the mission of several startups created in the last years, such as Qubole, just to mention one, the idea of monitoring and assessing analytics based on BDaaS was previously discussed by researchers and scholars [19], [24].

Finally, *Model-based BDaaS* is the closest line of research to the work in this paper. It considers approaches aimed to provide models for representing the different stages of a Big Data pipeline and improving the capability of verifying solutions against requirements. The high complexity and side-costs of designing, developing, and deploying Big Data infrastructures suggest the adoption of model-driven approaches that foster modularity, reusability, and automation of design and implementation tasks. MBDAaaS [7] aims to address the above needs. Recent contributions have been developed on top of platform-specific configuration libraries. KeystoneML [25], for example, introduced an approach for large-scale pipeline optimization extending Spark ML libraries [26]. The authors focus on capturing end-to-end pipeline application characteristics that are used to automatically optimize execution at both the operator and pipeline application levels. Other works, such as [27], have focused on testing and monitoring Machine Learning going beyond error rate but focusing on system reliability, that is, reducing technical debt and lowering long-term maintenance cost.

A high-level data-flow abstraction for modeling complex pipelines is also proposed in [28]. The data flows proposed in this work are direct acyclic graphs that specify some aspects of a pipeline delegating data inspections and optimization to the execution stage. Baylor et al. [29] propose an adaptation of TensorFlow for supporting data analysis, transformation and validation. The aim is boosting automation in the deployment of machine learning models. The main limitations of the current proposals are that they are closely tied to specific frameworks, such as Spark in [25], [28] or TensorFlow in [29], and lack of a formal definition supporting verification procedures for BDA pipelines.

Although the above perspectives have been considered in the literature, there is a lack of a comprehensive approach addressing the whole lifecycle of MBDAaaS. This paper considerably extends the work in [18] to provide a general and enhanced MBDAaaS methodology that includes: *i)* an extended declarative model formalizing requirements and constraints definition, including a verification procedure based on the concept of interference for the definition of consistent declarative models, *ii)* an extended procedural model defining an OWL-S service composition linked to specification in the declarative model. Furthermore, it defines a compiler that semi-automatically transforms an OWL-S Service Composition in a generic deployment model, then focusing on Oozie-based workflows. Finally, it proposes an extended architecture and its evaluation in a complex scenario.

8 Conclusions

Today, development of Big Data applications often takes a bottom-up, technology-driven approach, where the latest release of some technology drives the development of Big Data applications. In addition, the execution workflow of a Big Data application and corresponding computations are often hidden to Big Data developers and architects, as well as to the final users. In this paper, we proposed a complete methodology for MBDAaaS. Our approach proposes a *return to roots*, advocating a top-down approach to Big Data application development. Our approach is based on the specification of models supporting a semi-automatic definition and deployment of Big Data campaigns, followed by the definition of technology-independent scripts, and finally by the execution of a deployment process on available technologies. Also, the semantic modeling supported by our methodology formally represents all the details of a Big Data campaign, providing to developers and final users a direct understanding of Big Data computations and their behavior. The application of the methodology to a real-world scenario on SAP premises demonstrated its feasibility and the absence of obstacles to its deployment.

Acknowledgments

This work was partly supported by the European Union's Horizon 2020 research and innovation programme under the TOREADOR project, grant agreement No 688797, and by the programme "Piano sostegno alla ricerca 2015-17" funded by Università degli Studi di Milano.

References

[1] IDC, *Worldwide Semiannual Big Data and Analytics Spending Guide*, March 2017, http://www.idc.com/getdoc.jsp?containerId=prUS42371417.

[2] Y. Demchenko, P. Grosso, C. de Laat, and P. Membrey, "Addressing big data issues in scientific data infrastructure," in *Proc. of CTS 2013*, San Diego, CA, USA, May 2013.

[3] D. Agrawal, S. Das, and A. El Abbadi, "Big data and cloud computing: current state and future opportunities," in *Proc. of EDBT/ICDT 2011*, Uppsala, Sweden, March 2011.

[4] R. F. da Silva, R. Filgueira, I. Pietri, M. Jiang, R. Sakellariou, and E. Deelman, "A characterization of workflow management systems for extreme-scale applications," *Future Generation Computer Systems*, vol. 75, pp. 228–238, October 2017.

[5] V. Markl, "Breaking the chains: On declarative data analysis and data independence in the big data era," *VLDB Endowment*, vol. 7, no. 13, pp. 1730–1733, August 2014.

[6] S. Madden, "From databases to big data," *IEEE Internet Computing*, vol. 16, no. 3, pp. 4–6, 2012.

[7] C. Ardagna, P. Ceravolo, and E. Damiani, "Big data analytics as-a-service: Issues and challenges," in *Proc. of PSBD 2016*, Washington, VA, USA, December 2016.

[8] D. Martin, M. Paolucci, S. McIlraith, M. Burnstein, D. McDermott, D. McGuinness, B. Parsia, T. R. Payne, M. Sabou, M. Solanki *et al.*, "Bringing semantics to web services: The owl-s approach," 2004.

[9] Q. Z. Sheng, X. Qiao, A. V. Vasilakos, C. Szabo, S. Bourne, and X. Xu, "Web services composition: A decade's overview," *Information Sciences*, vol. 280, pp. 218–238, 2014.

[10] A. De Renzis, M. Garriga, A. Flores, A. Cechich, and A. Zunino, "Semantic-structural assessment scheme for integrability in service-oriented applications," in *Proc. of CLEI 2014*, Montevideo, Uruguay, September 2014.

[11] I. Hashem, I. Yaqoob, N. Anuar, S. Mokhtar, A. Gani, and S. Khan, "The rise of big data on cloud computing: Review and open research issues," *Information Systems*, vol. 47, pp. 98–115, 2015.

[12] D. Mysore, S. Khupat, and S. Jain, "Big data architecture and patterns, part1: Introduction to big data classification and architecture," *IBM Corp*, 2013.

[13] D. Terzi, R. Terzi, and S. Sagiroglu, "A survey on security and privacy issues in big data," in *Proc. of ICITST 2015*, London, UK, December 2015.

[14] WWW Consortium *et al.*, "JSON-LD 1.0: a JSON-based serialization for linked data," 2014.

[15] P. Bosc, E. Damiani, and M. Fugini, "Fuzzy service selection in a distributed object-oriented environment," *IEEE Transactions on Fuzzy Systems*, vol. 9, no. 5, pp. 682–698, Oct 2001.

[16] T. Mens and P. Van Gorp, "A taxonomy of model transformation," in *Proc. of GraMoT 2005*, Tallinn, Estonia, September 2005.

[17] A. Oprea, Z. Li, T.-F. Yen, S. Chin, and S. Alrwais, "Detection of early-stage enterprise infection by mining large-scale log data," in *Proc. of DNS 2015*, Rio de Janeiro, Brazil, June 2015.

[18] C. Ardagna, V. Bellandi, M. Bezzi, P. Ceravolo, E. Damiani, and C. Hebert, "A model-driven methodology for big data analytics-as-a-service," in *Proc. of BigData Congress 2017*, Honolulu, HI, USA, June 2017.

[19] M. D. Assunção, R. N. Calheiros, S. Bianchi, M. A. Netto, and R. Buyya, "Big data computing and clouds: Trends and future directions," *Journal of Parallel and Distributed Computing*, vol. 79, pp. 3–15, 2015.

[20] C. Ardagna, E. Damiani, F. Frati, and D. Rebeccani, "A configuration-independent score-based benchmark for distributed databases," *IEEE Transactions on Services Computing (TSC)*, vol. 9, no. 1, pp. 123–137, January 2016.

[21] Z. Zheng, J. Zhu, and M. R. Lyu, "Service-generated big data and big data-as-a-service: An overview," in *Proc. of the IEEE Congress on Big Data*, Santa Clara, CA, USA, June–July 2013.

[22] H. Herodotou, H. Lim, G. Luo, N. Borisov, L. Dong, F. Cetin, and S. Babu, "Starfish: A self-tuning system for big data analytics." in *Cidr*, vol. 11, no. 2011, 2011, pp. 261–272.

[23] G. Skourletopoulos, C. Mavromoustakis, P. Chatzimisios, G. Mastorakis, E. Pallis, and J. Batalla, "Towards the evaluation of a big data-as-a-service model: a decision theoretic approach," in *Proc. of INFOCOMW 2016*, San Francisco, USA, April 2016.

[24] M. M. Salvador, M. Budka, and B. Gabrys, "Adapting multicomponent predictive systems using hybrid adaptation strategies with auto-weka in process industry," in *Proc. of AutoML 2016*, New York, NY, USA, June 2016.

[25] E. R. Sparks, S. Venkataraman, T. Kaftan, M. J. Franklin, and B. Recht, "Keystoneml: Optimizing pipelines for large-scale advanced analytics," in *Proc. of ICDE 2017*, San Diego, CA, USA, April 2017.

[26] X. Meng, J. Bradley, B. Yavuz, E. Sparks, S. Venkataraman, D. Liu, J. Freeman, D. Tsai, M. Amde, S. Owen *et al.*, "Mllib: Machine learning in apache spark," *The Journal of Machine Learning Research*, vol. 17, no. 1, pp. 1235–1241, 2016.

[27] E. Breck, S. Cai, E. Nielsen, M. Salib, and D. Sculley, "The ml test score: A rubric for ml production readiness and technical debt reduction," in *Proc. of IEEE Big Data 2017*, Boston, MA, USA, December 2017.

[28] J.-H. Böse, V. Flunkert, J. Gasthaus, T. Januschowski, D. Lange, D. Salinas, S. Schelter, M. Seeger, and Y. Wang, "Probabilistic demand forecasting at scale," *VLDB Endowment*, vol. 10, no. 12, pp. 1694–1705, 2017.

[29] D. Baylor, E. Breck, H.-T. Cheng, N. Fiedel, C. Y. Foo, Z. Haque, S. Haykal, M. Ispir, V. Jain, L. Koc *et al.*, "Tfx: A tensorflow-based production-scale machine learning platform," in *Proc. of SIGKDD 2017*, Halifax, Canada, October 2017.

Claudio A. Ardagna is an Associate Professor at the Dipartimento di Informatica, Università degli Studi di Milano. His research interests are in the area of big data and cloud security and assurance. He is the recipient of the ERCIM STM WG 2009 Award for the Best Ph.D. Thesis on Security and Trust Management. He has coauthored the Springer book "Open Source Systems Security Certification". The URL for his web page is http://www.di.unimi.it/ardagna

Valerio Bellandi is an Assistant Professor at the Dipartimento di Informatica, Università degli Studi di Milano. His research interests are in the area of big data and network communications and image processing. On these topics, he has published several scientific papers. As a data scientist, he was involved in several international research projects. The URL for his web page is http://www.di.unimi.it/bellandi

Michele Bezzi is a Research Manager at SAP Security Research. He has 15+ years experience in industrial research. He has been contributing to several European projects, (e.g., TOREADOR, Assert4SOA, CoCoCloud, SecCord, Effects+, Primelife, TAS3, SpikeForce) and he has published 50+ referred papers and 6 patents, in various research areas: security, privacy, pervasive computing, neural networks, evolutionary models, complex systems.

Paolo Ceravolo is an Assistant Professor at the Dipartimento di Informatica, Università degli Studi di Milano. His research interests include Data Representation and Integration, Business Process Monitoring, Empirical Software Engineering. On these topics, he has published several scientific papers. As a data scientist, he was involved in several international research projects and in innovative startups. The URL for his web page is http://www.di.unimi.it/ceravolo

Ernesto Damiani is a Full Professor at the Università degli Studi di Milano, where he leads the SESAR research lab, and the leader of the Big Data Initiative at the EBTIC/Khalifa University in Abu Dhabi, UAE. He is the Principal Investigator of the H2020 TOREADOR project. He was a recipient of the Chester-Sall Award from the IEEE IES Society (2007). He was named ACM Distinguished Scientist (2008) and received the Stephen S. Yau Services Computing Award (2016).

Cedric Hebert is this guy convinced that security can be made simple. As a certified Infosec expert of the SAP Security Research team, his work ranges from secure design to offensive security. Some of his current and past projects include the kickoff of SAP Enterprise Threat Detection solution design and implementation, blockchain security and threat deception. His has an active role in the adoption of anonymization technologies in the context of Big Data analytics.