Big Data Assurance: An Approach Based on Service-Level Agreements

Claudio A. Ardagna[a], Ernesto Damiani[a,b], Cedric Hebert[c], Maria Krotsiani, Christos Kloukinas, George Spanoudakis[d],

[a] *DSRC, DI, Università degli Studi di Milano*
Milano, 20133, Italy
[b] *AIISI, Khalifa University*
Abu Dhabi, UAE
[c] *SAP Labs France*
Mougins, 06250, France
[d] *City, University of London*
Northampton Square, London, EC1V 0HB, UK

Abstract

Big data management is a key enabling factor for enterprises that want to compete in the global market. Data coming from enterprise production processes, if properly analyzed, can support a boost in the enterprise management and optimization, guaranteeing faster processes, better customer management, and lower overheads/costs. Guaranteeing a proper Big Data analytics is the Holy Grail of Big Data, often opposed by the difficulty of evaluating the precision of the Big Data analytics results. This problem is even worse when Big Data analytics are provided as a service in the cloud, and must comply with both users' requirements and laws. Recently, the Big Data community has started noticing that there is the need to complete Big Data with assurance techniques proving the correct behavior of Big Data analytics and management. This paper provides an assurance solution based on Service-Level Agreements (SLAs), where stochastic formal models are used to define, negotiate, and monitor SLAs targeting the behavior of Big Data platforms and analytics delivered as a service.

Email addresses: claudio.ardagna@unimi.it (Claudio A. Ardagna), ernesto.damiani@ku.ac.ae (Ernesto Damiani), cedric.hebert@sap.com (Cedric Hebert), {Maria.Krotsiani, C.Kloukinas, G.E.Spanoudakis}@city.ac.uk (Maria Krotsiani, Christos Kloukinas, George Spanoudakis)

Keywords: Service-Level Agreements, Big Data Analytics, Big Data Assurance, Model Checking

1. Introduction

Big Data is a major research topic, leading all productive environments and enterprises towards the data-driven economy. Big Data becomes fundamental for all enterprises, from big ones to SMEs, which want to compete in the global market. Better management of data coming from productive processes in fact leads to faster processes, better customer management, and lower overheads/costs. Often, the immense benefits of Big Data are forbidden to those enterprises that do not have in-house Big Data skills and competences, and are therefore unable to manage the intrinsic complexity of its technologies.

In the last few years, the R&D community has focused on widening the adoption of Big Data technologies by providing solutions supporting users in easily implementing a Big Data campaign [1, 2]. Approaches to Big Data-as-a-Service have been defined, providing services for the management and execution of Big Data computations at different layers of the cloud stack. Substantial work has been done on Big Data Platform-as-a-Service, where users are provided with a pre-configured platform, as for instance Microsoft Azure HDInsight and Amazon EMR. Users need then to only concentrate on configuring and executing the analytics without worrying about how to manage and deploy the corresponding platform. This scenario, however, collides with the lack of data scientists having the skills and competences to implement a sound Big Data campaign and retrieve meaningful results. According to the Harvard Business Review [3], shortage in data scientist dates back to 2012, with an inexorable, increasing trend in data professional demand towards our days (IBM predicts job demand for data scientist, data developers, and data engineers to reach nearly 700,000 openings by 2020 with an increase of 28% [4]).

Following this issue, in the last few years, different techniques supporting the concept of Big Data Analytics-as-a-Service has been defined [5, 6, 7, 8, 9], where high-level requirements of the users are transformed in Big Data workflows that can be executed on the target Big Data platform. However, these approaches suffer from the inability to evaluate and manage the quality and correctness of the implemented Big Data analytics. It is therefore increasingly

important to guarantee that the overall Big Data infrastructure complies with users' expectation, and even more with national/international laws and regulations. This challenge points to the concept of *Big Data assurance*, which aims to provide justifiable confidence that a system behaves as expected. In the past, assurance techniques (*i.e.*, audit, certification, compliance) have been used to evaluate the status of distributed systems such as the cloud [10]. These solutions, however, barely apply to Big Data scenarios and require a rethinking of the assurance concept at large.

Effectively tackling such issues is fundamental to increase trust in Big Data-as-a-Service paradigm, and in turn to foster the movement of critical businesses to it. In this paper, we extend the work presented in [11] and provide an approach to Big Data assurance, whereby Service-Level Agreements (SLAs) help evaluating the compliance of a Big Data environment and corresponding analytics, both distributed as a service, to (security) properties. We consider a Model-based Big Data Analytics-as-a-Service (MBDAaaS) methodology (Section 3), where the users specify a set of declarative requirements, driving the deployment of the Big Data services to accomplish specific analytics. After introducing the concept of Big Data assurance, discussing data, process, and platform assurance (Section 4), we present our approach to Big Data assurance (Section 5), where SLAs are defined according to declarative requirements of the users and configurations of the target execution platform. Moreover, we present different SLA negotiation process (Section 6), which is supported by the PRISM [12] probabilistic model checker, and our solution to SLA monitoring (Section 7). The performance of the proposed approach is finally evaluated in a simulated environment (Section 8) proposed by SAP, a global market leader in enterprise resource planning.

To conclude, the approach in this paper develops on the approach presented in [11] providing the following contributions *i)* a refined assurance methodology, which mixes assurance requirements and controls at declarative and procedural levels using annotations functions, *ii)* two types of SLA negotiation processes, namely client-middleware-server negotiation and client-server negotiation, *iii)* a new approach to SLA monitoring.

2. Related Work

Non-functional assurance verification can be defined as the way to gain justifiable confidence that a system will consistently demonstrate one or more

properties, and operationally behave as expected despite failures and attacks [13]. As discussed in [10], non-functional assurance consists of methodologies and techniques (e.g., audit, certification, and compliance) for collecting and validating evidence supporting different security properties, such as, security, privacy, and dependability.

Several assurance techniques have been defined in the past for distributed systems, from service-oriented architectures (*e.g.*, [14, 15]) to cloud environments [16, 17, 18, 19, 20, 21]. These techniques mostly focused on evaluating the security and dependability status of distributed systems. More specifically, existing processes evaluate the status of the system at all layers of the corresponding stack (*e.g.*, service, platform, and infrastructure layers for the cloud), provide a process that continuously evaluates the correctness of the system behavior, and support incremental evaluation to reduce the need of evaluating the system from scratch at each contextual change. Lot of research has also been done to assess the behavior of Big Data technologies. Several approaches have been developed in the area of SLA for Big Data applications, mainly targeting MapReduce jobs [22, 23, 24] and Hadoop [25], where they either tried to identify and address SLA violations or found appropriate algorithms taking into consideration the performance of these services at all cloud layers [26].

In this paper, SLAs are used as a means for supporting Big Data assurance. SLAs have been heavily used to specify terms and conditions for service provision between service consumers and service providers. To support the SLA specification process, several specification languages (*e.g.*, [27, 28, 29, 30, 31]) have been defined over the years, with the aim to simplify the SLA specification process for the involved parties, and to minimize the time and cost required for it. Despite the extended research on SLA languages and SLA management, SLAs still fail to completely address the requirements of BDA services. Even though new SLA languages have been developed to support cloud services (*e.g.*, CSLA [32], SLAC [33], SLA* [30]), security abd privacy of services is not fully supported. Finally, cloud and BDA services support complex operations and have many dependencies between different operations/services, which are difficult to model with current SLA languages.

3. MBDAaaS Methodology

Big Data analytics and management are practical pressing needs, which represent turning key aspects for enterprises that want to compete in the

global market. While a Big Data computation can be easily deployed, it still takes substantial expert knowledge to set the computation up in such a way that it actually produces meaningful and sound results. Current approaches mostly focused on providing the so-called Big Data Platform-as-a-Service (BDPaaS) paradigm, where Big Data providers (*e.g.*, Azure, Amazon) offer Big Data platforms on demand to end users. End users then access a complete Big Data platform, without the need of knowing how to install and configure it, and just focus on implementing a Big Data computation. BDPaaS, though important, is not sufficient to widen the adoption of Big Data technologies among the different production domains. It is fundamental to support the users in the management of Big Data analytics and all involved activities.

Recently, the research community moved from a paradigm where Big Data application development is driven by the latest technology release, to a more traditional (model-based) paradigm where the users specify their expectations in terms of requirements, and users/consultants follow them in implementing the Big Data computation. In [5], we presented a Model-based Big Data Analytics-as-a-Service (MBDAaaS) approach, where users specify declarative requirements driving smarter components in carrying out the Big Data processes. MBDAaaS is based on the Model Driven Engineering (MDE) paradigm and aims to reduce as much as possible the involvement of the users in Big Data management.

In the following, after describing our reference scenario, we briefly summarize the approach in [5].

3.1. Reference Scenario

We use a reference scenario provided by SAP, a global market leader in enterprise resource planning, where an ERP system continuously sends its application logs to two connected cloud platforms, a private cloud Monsoon instance and a publicly accessible AWS instance saving data into a HADOOP data store (Figure 1). In the private cloud, the data are first pseudonymized and then anonymized with differential privacy before being streamed to the AWS instance. There, the data are constantly monitored by a Retention Service, configured to delete logs older than 1 year. An auditor (Clara) can connect to this service and manually check whether there is a violation of the retention policy. An analyst (Bob) can check the logs through an analytics agent, which first clusters logs and then further anonymize big clusters with k-anonymity. k-anonymity is configured to run only on 'big' clusters. Small clusters are considered anomalies and not eligible to k-anonymity protection.

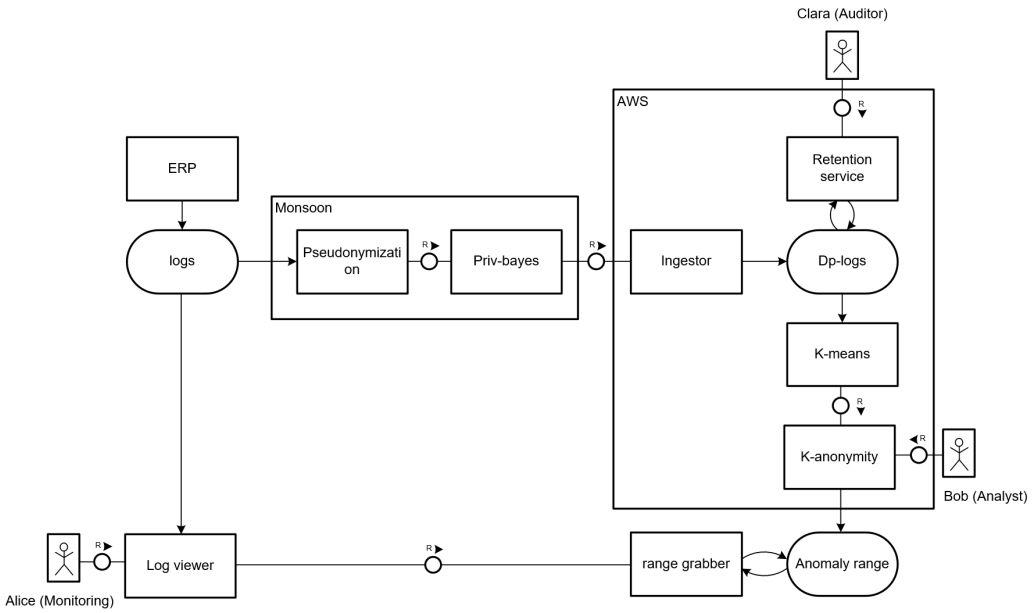


Figure 1: Architecture diagram

Once his analysis is complete, Bob defines anomaly ranges which allows the response team (Alice) to look at the raw logs whitelisted by Bob, enforcing a 4-eyes principle. Alice uses a log viewer for taking the proper decision, such as contacting the identified fraudster for a face-to-face interview. When using the log viewer, the raw, non-anonymized logs are displayed to her, but only if they pertain to the ranges defines in the Anomaly Range storage by Bob (e.g., specific log IDs, specific time-ranges).

3.2. MBDAaaS Models and Workflow

The proposal in this paper is based on the approach in [5], which builds on three models described below.

Declarative model. It collects a user's requirements and expectations on the Big Data computation. The user accesses a graphical user interface and defines such requirements in terms of *goals* modeled as pairs (*indicator*, *objective*). An indicator permits to measure the goal, while the objective represents a value the indicator must achieve. Each goal can also be enriched with constraints as pairs (*attribute*, *value*), further refining user's expectations. It is also coupled with a priority that permits to solve conflicts in

the user's specifications and support the user in defining sound declarative models. It is important to note that, unlike existing solutions mostly focusing on data modeling and representation [34], the approach in [5] permits to define declarative requirements along all phases of a Big Data computation including: *i)* data preparation, *ii)* data representation, *iii)* data analytics, *iv)* data processing, and *v)* data visualization and reporting. The declarative model is specified as a JSON file and represents the input of the MBDAaaS methodology.

Procedural model. It is a technology-independent model and defines the workflow describing how the Big Data processes should be carried out to achieve the objectives. It is modeled as a direct acyclic graph where *i)* each node represents a service in one of the five phases of declarative model specification, *ii)* each arrow between two nodes represents the execution flow. Each service can be configured according to constraints in the declarative model or preferences directly specified by the user. The procedural model defines a service composition specified in a semantic language based on OWL-S [35].

Deployment model. It is a technology-dependent model that specifies how the Big Data computation is executed on the target platform. It is a technology-dependent representation of the service composition in the procedural model, using a workflow language (*e.g.*, Oozie, Spring Cloud Dataflow) that can be automatically executed on the target platform. A compiler is used to transform the OWL-S procedural model in a language-specific workflow. The deployment model contains all details on the target system and implemented algorithms, and can be used as a source of assurance requirements.

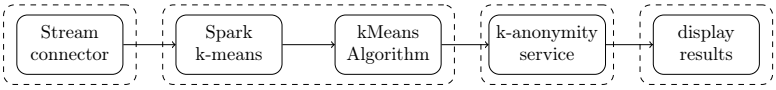
Example. Figure 2 presents an example of declarative, procedural, and deployment models on our reference scenario in Section 3.1. The application data logs entail sensitive information on the actions performed by the employees. SAP can then define a declarative model, whose JSON representation is presented in Figure 2, with the following goals:

1. goal *Anonymization* for the data preparation phase, with indicator *Anonymization technique* and value *Priv-bayes* [36], extended with a constraint on the noise level such as (*Algorithm*, *epsilon=0.1*);

Declarative Model (Excerpt)


```
{
[...]
"tdm:targetDataSources": [
    "hdfs://192.168.0.5:8020"
],
[...]
"tdm:preparation": {
[...]
"@type": "tdm:Area",
    "tdm:label": "Data Preparation",
    "tdm:incorporates": [
    {
        "@type": "tdm:Goal",
        "tdm:label": "Anonymization",
        "tdm:constraint": "{}",
        "tdm:incorporates": [
         {
            "@type": "tdm:Indicator",
            "tdm:label": "Anonymization Techniques",
            "tdm:constraint": "{}",
            "tdm:visualisationType": "Option",
            "tdm:incorporates": [
              {
                "@type": "tdm:Objective",
                "tdm:constraint": "{k:10}",
                "tdm:label": "K−Anonymity"
              }]}]}],
[...]
]}
```


Procedural Model



Deployment model (SCDF)

```
PSEUDO: pseudonymization−service
−−parameter.output_column=pseudo
−−parameter.input_column=user
−−parameter.input_file=hdfs://user/root/sap/anon1/input.csv
−−parameter.input_data=hdfs://user/root/sap/anon1/input_data.csv
−−parameter.delimiter=;
−−parameter.output_file=hdfs://user/root/sap/anon1/output.csv
&& PRIV: privbayes−service
−−input_file=hdfs://user/root/sap/anon1/output.csv
−−delimiter=;
−−output_file=hdfs://user/root/sap/anon1/results.csv\\
```

Figure 2: An example of model specification

2. goal *Anonymization* for the data preparation phase, with indicator *Anonymization technique* and value *Pseudonymization*, extended with a constraint such as (*Algorithm*, *hash=SHA-256*);
3. goal *Analytics Aim* for the data analytics phase, with *i)* indicator *Task* and value *Crisp Clustering* and *ii)* indicator *Learning Approach* and value *Unsupervised*.
4. goal *Compliance* for the data analytics phase, with indicator *Data Erasure* and value *1 year*;
5. goal *Anonymization* for the data processing phase, with indicator *Anonymization technique* and value *K-anonymity*, extended with constraints such as (*Algorithm*, *K=10*) and (*Exception*, *Cluster-size<5*);

A procedural model for the AWS instance then applies *i)* a stream data connector to ingest SAP data on application logs, *ii)* an analytics service clustering logs based on their similarity, and *iii)* an anonymization service generalizing the content of each cluster before displaying it to the analyst. The deployment model finally specifies the details (including the endpoints and using Spring Cloud DataFlow language) of the anonymization and analytics services composed as described in the procedural model.

The MBDAaaS methodology is then composed of a series of semi-automatic model transformations. It takes as input the declarative model and produces as output the deployment model that is executed on the target platform. The declarative model is first defined by the user. On the basis of the defined goals and constraints, our methodology retrieves a set of services that are compatible with the user goals. For instance, considering Goal 1 *Anonymization* above, our methodology retrieves all services implementing a priv-bayes anonymization service. Upon retrieving all services, the user manually configures and composes a subset of them to produce the procedural model. This model is then automatically transformed by a compiler in a deployment model, which is ready to be executed on the target platform [5].

4. Big Data Assurance

The success of a methodology that assists users in instantiating their Big Data computation depends on the ability of guaranteeing that the retrieved results are in line with the user expectations. In our case, the retrieved results must be the results of a process correctly implementing the user's declarative specifications. The soundness of the entire process depends on whether the

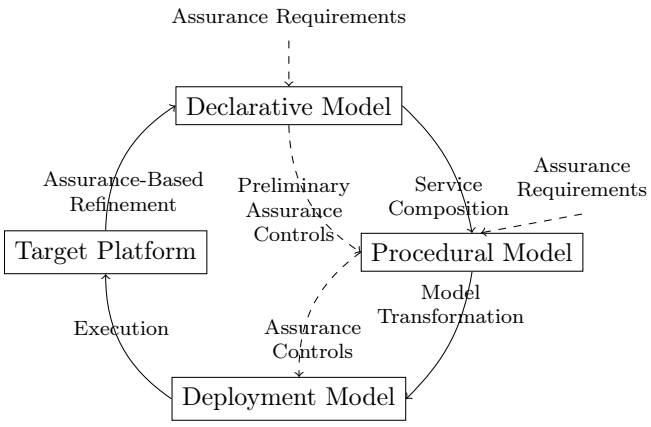


Figure 3: Model-based Big Data Analytics-as-a-Service methodology

Big Data computation satisfies the assurance requirements specified by the user. Declarative requirements and procedural specifications become then sources of assurance requirements that need to be verified both at run time and a posteriori.

4.1. Assurance Methodology and Requirements

Big Data assurance extends traditional assurance to embrace different layers of evaluation. In addition to the evaluation of the status of the platform and its behavior, Big Data assurance process must evaluate the status of the analytics process, as well as the status of the corresponding data. The latter points to *veracity* of the 5V model for Big Data [37], which refers to the trustworthiness of data.

In this paper, we focus on the assurance evaluation of our MBDAaaS methodology. Figure 3 sketches how an assurance process aimed to verify the compliance to specified requirements can be integrated with our methodology. Specifically, the declarative and procedural models in Section 3 are extended with two annotations specifying assurance requirements. Assurance requirements are first associated with specifications at declarative level, which in turn can also be a direct source of assurance requirements. For instance, a declarative specification on the expected data anonymity technique can directly point to an assurance requirement checking the adopted anonymization technique. Declarative requirements are then integrated and refined with assurance requirements specified at procedural level. The latter are associated with specific services and monitor their computation. Once

the extended procedural model is fed into the compiler a deployment model is produced, where each service is associated with the set of assurance controls monitoring their execution and evaluating their compliance against assurance requirements. Each service in the deployment model is then executed together with corresponding assurance controls on the *target platform*. The target platform includes a set of assurance managers, each responsible to execute a subset of assurance controls for a specific service. The retrieved assurance evaluation is then used to refine declarative model specification towards full compliance with the user's expectation.

MBDAaaS assurance must be evaluated at three layers that can be linked to the five areas of declarative model specification, namely, data preparation, data representation, data analytics, data processing, data visualization and reporting, as follows.

- *Data assurance.* It evaluates how data are managed, represented, and prepared for the Big Data platform. It relates to the two areas of the declarative model regarding data preparation and data representation. Data preparation consists of all activities that need to be done at data collection/ingestion time to prepare data for analytics (e.g., cleansing and anonymization). Data representation consists of the choices made for representing data to be analyzed (e.g., data model, data structure, and data management). In our scenario, the analyst must never see individual data records, but only data buckets generalized by applying k-anonymity. An assurance manager then verifies that each displayed data record cannot be distinguished from at least $k-1$ other records. As another example, the response team person must only be able to access data records previously marked as suspicious by the analyst. An assurance manager then verifies that the records made available match with the specified anomaly ranges.

- *Process assurance.* It evaluates the status of a Big Data process, by monitoring all activities between the ingestion of data and the production of the computation outcome. It mainly relates to the two areas of declarative model regarding data analytics, and data visualization and reporting. Data analytics specifies the mining operations to be executed (e.g., the type and the model of analytics, the target quality). Data visualization and reporting specifies how the outcome of the computation must be presented (e.g., information on the dimensionality, cardinality, and graph type). In our scenario, data must not be kept

Table 1: Mapping between declarative areas and assurance layers

Assurance Layer	**MBDAaaS Areas**	**Description**
Data	Data preparation, Data representation	Evaluate the properties related to data, such as their anonymity, privacy, format.
Process	Data analytics, Data visualization & reporting	Evaluate the Big Data process properties, monitoring the run-time execution of analytics, the results and their visualization.
Platform	Data processing	Evaluate the platform status: deployed components, their configuration and availability, and adherence to standards and guidelines.

on the cloud for more than one year. To this aim, a deletion service runs every few minutes on the target system and an assurance manager verifies whether data records are never older than 1 year. As another example, data must be manipulated, stored, and accessed in an European country following General Data Protection Regulation (GDPR). This can be achieved by monitoring the whole process, verifying the location of the machine where data are managed.

- *Platform assurance.* It evaluates the status of the Big Data platform, where computations are executed. It mainly considers the status of those components and algorithms that are concerned with data and process assurance. It relates to and goes beyond the data processing area of the declarative model, concerned with data routing and parallelization. It also considers non-functional components supporting data security and privacy, and process monitoring. For instance, it includes requirements on performance, parallelization, processing, and elasticity. It permits to verify that a given non-functional (*e.g.*, signature component for data integrity) component is behaving as expected. In our scenario, the Big Data platform must guarantee data integrity and a given level of availability. Assurance manager then verifies that the records are cryptographically signed and the signature is valid. It also monitors the entire platform evaluating its availability.

Table 1 summarizes the mapping between the 5 areas of declarative model and the three layers of assurance.

4.2. Assurance Requirements Specification

With a very general setting (Figure 3), we assume assurance requirements on Big Data computations to be specified in terms of assurance annotations of goals in the declarative model or services in the procedural model. The only assumption we introduce on such annotations is the existence of a (semi-automatic) means to translate these requirements in real assurance controls at deployment level, such that, services are executed together with the controls responsible for their evaluation. Assurance annotations are defined as follows.

Definition 4.1 (Assurance Requirements). *Assurance requirements* Λ *are a set of assurance annotations* λ *in the form (*attr op value*), where* attr *is an assurance target,* op *is an operator in* $\{=,<,>,\neq,\leq,\geq\}$*, and* value *the objective for the attribute.*

Our generic definition of assurance requirements permits to capture different ways of expressing assurance annotations, as well as different target levels. For instance, assurance annotations can be defined using different techniques from SLAs (as in our paper) to certification, audit, and compliance. Regarding their specification, assurance annotations could target existing standards and regulations (e.g., GDPR), point to generic aspects and configurations of Big Data services, or pose restrictions on the way in which specific declaration can be achieved.

Assurance annotations can be specified at two different layers: declarative specifications d in the declarative model and component service s in the procedural model. Let us discuss the semantics of the annotation function λ as follows.

- $\lambda(d)$. It defines assurance annotations on specific elements of the declarative model. Recalling that a declarative specification d is triple (G, I, O), where G defines the goal to be achieved by the computation, I is an indicator that represents a way of measuring or assessing the goal, and O is an objective that represents the target to be achieved to consider the goal fulfilled, $\lambda(d)$ associates an assurance constraint with a specific d. For instance, declarative specification d_a=(**Anonymization**,`Anonymization_Technique`, k-anonymity) requires the adoption of a k-anonymity technique to prepare data. This specification can be extended with an annotation $\lambda(d)$ that points to a

specific regulation (e.g., GDPR). This regulation might prescribe some requirements to be checked by our assurance methodology (e.g., the value of k) to retrieve a positive evaluation. It is important to note that declarative specifications can become a source of assurance requirements themselves. For instance, declarative specification d_a requires an assurance control checking the correct working of the k-anonymity technique.

- $\lambda(s)$. It defines assurance annotations on specific services in the procedural model. These annotations define those checks that directly transform in controls and are fed to the assurance monitor deployed within the same environment where services s are executing. For instance, a service s implementing a k-means clustering algorithm can be annotated with an assurance requirement prescribing locality of computation, that is, all computation should be executed in a specific country.

Annotations $\lambda(d)$ and $\lambda(s)$ are integrated and compiled to form a deployment model that *i)* includes a ready-to-be-executed workflow of the computation and *ii)* integrates all assurance controls devoted to the evaluation of the compliance between the computations and the user's expectations. We note that, as shown in Figure 3, the declarative and procedural models are the main sources of requirements and, in turn, the targets of assurance specification and verification. In the remaining of the paper, we show how a Big Data assurance process for our MBDAaaS methodology can be defined based on SLA specification and management. Annotations are used to negotiate the final SLAs to be monitored, which evaluate the adherence of the implemented Big Data processes to the user's expectations.

5. SLA Specification

SLAs are formal agreements (aka contracts) that clarify the service provisioning and the responsibilities between the service consumer and the service provider, to facilitate the communication between them [38]. At a minimal level, SLAs should define the security properties that need to be preserved during the provision of a service and the penalties that will be applied, in case the service provision violates those security properties.

Our SLA specification language is based on the WS-Agreement [27] extended to specify SLAs that are suitable to cover the MBDAaaS concept.

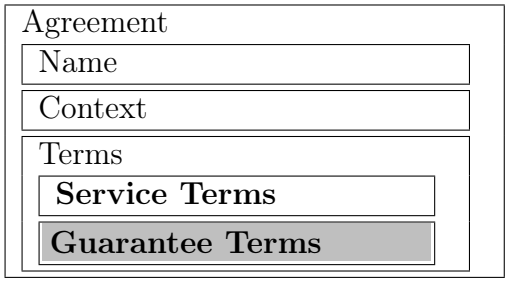


Figure 4: Structure of the WS-Agreement SLA

The WS-Agreement specification language was introduced by the Open Grid Forum to address some key requirements for the specification of SLAs, such as supporting modularity, accommodating other external and domain specific standards, and allowing extensions.

In WS-Agreement, an SLA is composed of three main sections, that is, the name, the context and the terms that are applied, as shown in Figure 4. The first section provides an optional SLA name. The second section, called *Context*, contains the metadata for the entire SLA (*e.g.*, the SLA participants, its lifetime). The third section specifies the terms of the SLA, which can be of two types: *a)* Service Terms that describe the services regulated by the SLA and *b)* Guarantee Terms (GTs) that specify the service levels that should be satisfied during the provision of the service. More specifically, GTs define the expected quality of service (QoS), comprising *i)* the obligated party, *ii)* the list of services this guarantee applies to, *iii)* an optional condition that must be met for a guarantee to be enforced, *iv)* an assertion expressed over service descriptions (*ServiceLevelObjective*), and *v)* one or more business values associated with this objective (*BusinessValueList*).

The limitations of WS-Agreement for our purposes in Section 4 are related to the lack of specification support for: *i)* security and privacy Service Level Objectives (*SLOs*); *ii)* actions that need to be taken during the life cycle of an SLA (*e.g.*, service provision platform modification actions or SLA renegotiation actions); *iii)* multi-party SLAs; and *iv)* comprehensive models of complex services, which include internal operations, service assets, data and other dependencies that Big Data Analytics services usually have.

We then extend WS-Agreement to support monitorable SLOs, on one side, and the specification of actions that should be undertaken in case of guaranteed term violations, on the other side. The BNF grammar of the ex-

Listing 1: Extended WS-Agreement BNF grammar (abstracted)

```
XWSAgreement ::= { IdOpt ... Terms ... }
Terms ::= { ... { GuaranteeTerms } }
GuaranteeTerms ::= List{GuaranteeTerm, ""}
GuaranteeTerm ::= { Id ... BusinessValueListType }
BusinessValueListType ::= ... CustomBusinessValue
CustomBusinessValue ::= List{XAction, ""}

XAction ::= "renegotiate" Pred
          | "penalty" Pred Int
      |  Id Pred
Pred ::= "true" | "false"
          | "(" Pred ")" [bracket]
      | Pred "&" Pred | Pred "|" Pred
      | "!" Pred | Pred "?" Pred ":" Pred
      | NumExp "=" NumExp | NumExp "!=" NumExp
      | NumExp ">" NumExp | NumExp "<=" NumExp
      | NumExp "<" NumExp | NumExp ">=" NumExp
NumExp ::= Int  |  "(" NumExp ")" [bracket]
      | NumExp "*" NumExp | NumExp "/" NumExp
      | NumExp "+" NumExp | NumExp "-" NumExp
      | Pred "?" NumExp ":" NumExp
      | "violations" "(" Id ")"
      | "penalty_amount" "(" Id ")"
      | "counter" "(" Id "," Id ")"
```

tended WS-Agreement is shown in Listing 1. The grammar syntax is based on the syntax used by the K Framework [39]. More specifically terminals are inside quotes, and *List{ X, "c" }* is a *c*-separated list of zero or more *X*s. Notes inside square brackets ([]) on the right have to do with evaluation. *seqstrict* means that they must be evaluated left-to-right, and *strict(1)* means that only the first non-terminal needs to be evaluated. Both *Pred* and *NumExp* permit the ternary if-then-else operator *pred ? exp1 : exp2*, which evaluates to *exp1* when *pred* is true, and to *exp2* otherwise. Apart from constants, one can use predefined functions, such as **violations**(GT), **penalty_amount**(GT), and **counter**(Action, GT). These functions provide the number of times a specific GT has been violated so far; the amount accrued in penalties due to violations of a specific GT; and the number of times an action has been executed, following a GT triggering, respectively.

More in detail, to support monitorable SLOs, we extended the subelement *CustomServiceLevel* of the *ServiceLevelObjective*, by introducing a new type for *CustomServiceLevel* called *PreciseSLOType*, to support the specification of SLOs at two levels.

1. Declarative level, based on the declarative model of our MBDAaaS approach (Section 3). This is specified as a property of a particular

category that is applied to a service asset (*e.g.*, internal or external operation or data elements of the service).

2. Procedural-Deployment level, which provides the exact run-time assertion representing the SLO satisfaction, to support a run-time monitoring process of the GTs. SLO assertions are expressed in the language developed in the CUMULUS project [40] — itself based on Event Calculus [41], a formal first order temporal logic language, also used by the EVEREST Framework [42]

As an example, according to our reference scenario and the goals of the declarative model, we can extract the information regarding security property *Pseudonymization*, which should be guaranteed for data in application logs following the specified algorithm (*e.g.*, SHA-256). Based on the corresponding procedural and deployment models, we can then retrieve the service to which the considered property applies.

Moreover, to support the specification of actions that should be undertaken when guaranteed terms are violated, we extend the WS-Agreement's sub-element *CustomBusinessValue* of element *BusinessValueList*. Our extension permits to express different types of actions that should be triggered by each GT violation, or by any other conditions specified in an SLA. Two predefined actions exist: *i) renegotiate* Pred, which causes the SLA to be renegotiated when the guard Pred is satisfied and *ii) penalty* Pred Int, which causes a penalty (or reward if negative) to be incurred. Even though WS-Agreement supports the specification of penalties, our extension makes a more precise penalty specification. SLA modellers are also free to use any other action name they wish and can also guard all actions with a predicate. A *guard* predicate declares conditions that should be satisfied, in addition to the GT violations triggering the action.

6. SLA Negotiation

SLA negotiation is characterized by the observation that a customer and a service provider are separated by a knowledge gap when initiating the SLA negotiation process. This knowledge gap may be due to lack of experience, lack of knowledge about the organizational structure, and lack of general knowledge with respect to the service, the service environment, or the deployment of the service, as follows: *i) general knowledge* about a used software; *ii) deployment-dependent knowledge* about the use, development,

Listing 2: Jess Rule

```
<tnsr:NegotiationRule name="rule1">
   <tnsr:If>
      <tnsr:LogicalExpression>
         <slac:Condition relation="LESS-THAN">
            <slac:Arg1><slac:QualityAttribute
             name="PENALTY-AMOUNT" party="CONSUMER"/>
            </slac:Arg1>
            <slac:Arg2><slac:Constant
             type="NUMERICAL" unit="EUR"> 100
              </slac:Constant></slac:Arg2>
         </slac:Condition>
         <slac:LogicalOperator> AND </slac:LogicalOperator>
         <slac:Condition relation="EQUALS">
            <slac:Arg1><slac:QualityAttribute
              name="PSEUDONYMIZATION" party="CONSUMER"
              unit="SHA-256"/>
            </slac:Arg1>
            <slac:Arg2><slac:Constant
               type="BOOLEAN">TRUE</slac:Constant>
            </slac:Arg2>
         </slac:Condition>
      </tnsr:LogicalExpression>
   </tnsr:If>
   <tnsr:Then>
      <tnsr:Action>
         <tnsr:Accept>
            <tnsr:QualityAttribute
               name="PENALTY-AMOUNT" party="CONSUMER"/>
            <tnsr:QualityAttribute
               name="PSEUDONYMIZATION" party="CONSUMER"/>
         </tnsr:Accept>
      </tnsr:Action>
   </tnsr:Then>
</tnsr:NegotiationRule>
```

or testing of the service; *iii) organization-dependent knowledge* about the way the software fits into the enterprise structure [43]. By decreasing this knowledge gap, customers and service providers can successfully negotiate a reasonable SLA for a specific service provision.

6.1. SLA Negotiation Framework

Different negotiation frameworks have been developed so far, each of which has different characteristics and can support different functionalities. Our approach is based on the PROSDIN negotiation framework, a proactive run-time SLA negotiation tool ([44]). The SLA negotiation process in PROSDIN is executed by a component called negotiation broker and according to specific negotiation rules. The negotiation broker manages the negotiation process on behalf of the interested parties, by providing access to different negotiation engines and by translating negotiation rules into the different negotiation specifications, accepted by these engines.

Our negotiation rules are Jess rules ([45]) that are defined in XML. These

rules are condition-action rules of the form:

$$\textbf{IF}(condition)\textbf{THEN}(action)\textbf{ELSE}(action)$$

Conditions are either atomic conditions or logical combinations of atomic conditions over specific property attributes of the relevant services. Rule actions can be of three types: *i) accept* actions that are used to accept the value of one or more property attributes in a given SLA offer; *ii) reject* actions that are used to reject the value of one or more property attributes in a given SLA offer; and *iii) set* actions that are used to propose a new value or range of values for one or more property attributes as part of an SLA offer. Our approach is similar to the one described above, as we are using Jess rules ([45]) for *accepting* or *rejecting* an SLA offer, or for *setting* new values to an offer. An example of a Jess negotiation rule is shown in Listing 2.

This rule states that if a consumer of a service has made an offer (or counter-offer) where the overall price to be paid for penalties is less than 100 euros and an SHA-256 encryption mechanism is used to pseudonymize data, then the offered values can be accepted.

6.2. SLA Negotiation Process

We define two types of SLA negotiation processes, varying the role of the MBDAaaS platform: *i) Client-Middleware-Server Negotiation*, where the MBDAaaS platform works as middleman between the client and the available service providers (*e.g.*, AWS), *ii) Client-Server Negotiation*, where the MBDAaaS platform plays the role of the service provider. In the first case, two different negotiation process can be implemented: *i)* the MBDAaaS platform performs a simple matching of the terms between the new SLA offer and existing SLAs from the other involved party (*i.e.*, a new client provides and SLA offer and a simple match of its terms is performed with existing service providers' SLA terms); *ii)* a negotiation process needs to be performed (*i.e.*, to provide counteroffers to change the terms of the offer and match existing SLAs) between the platform and the involved parties.

Client-Middleware-Server Negotiation. It takes the viewpoint of a middleman (*i.e.*, the MBDAaaS platform) between service consumers and service providers. When negotiating an SLA, there are two main types of SLAs we consider: *(i)* pay–as–you–go (PAYG) and *(ii)* subscription. They both assume an average rate of service use (what is known as "fair–use policies" in subscription contracts) and differ on how a service is charged. In PAYG,

the consumer pays a price each time a service is used, while, in subscription, they pay a subscription each interval (e.g., week, month). For conciseness, we consider PAYG SLAs only in two scenarios: when we can reach to an agreement between the involved parties by simply matching the terms (assurance requirements Λ in Section 4.2) and when some terms cannot be matched and their values need to be adjusted.

- *Simple matching at middleware side.* The MBDAaaS platform gets the terms (assurance requirements Λ in Section 4.2) of the offer made by the relevant party (service consumer or service provider) and tries to match them with the existing SLA terms from other parties, which have already signed SLAs with the platform. Thus, the Jess rules of the offer are generated and checked based on the process described in the previous section. However, in this case action *set* would not be applied, as a simple matching is performed without providing a counter-offer to change the terms. This is the simplest type of SLA negotiation process that can be performed by the MBDAaaS platform.

- *Negotiation between middleware and the relevant party.* The MBDAaaS platform tries to negotiate the terms (assurance requirements Λ in Section 4.2) of the offer made by one party in case they do not match existing ones from other parties. Thus, action *set* of the Jess rules is performed to match the terms that cannot be matched; if not possible, the offer is rejected, and a counter-offer to be agreed by the interested party sent back.

Client-Server Negotiation. The MBDAaaS platform provides the Big Data-Analytics services required by a client. It is the simplest type of negotiation as no service providers are involved. Thus, it just checks if the terms (assurance requirements Λ in Section 4.2) are convenient and supported, and respond backs whether the offer is accepted/rejected or with a counter-offer.

In all the above cases, we assume that for each consumer c_i we know the:

- C_i^{in}: income received per request;

- P_i^{in}: penalty paid when a request is not serviced;

- ${}^{C}R_i^{in}$: contractually agreed/negotiated average rate of service requests; and

- ${}^{O}R_i^{in}$: its observed/assumed average request rate.

Thus, for a new consumer, we might negotiate for a rate ${}^{C}R_i^{in}$, when we actually assume that they only have a lesser rate ${}^{O}R_i^{in}$. Similarly, for a consumer we have an SLA with, we would have agreed upon a ${}^{C}R_i^{in}$ but observe that she actually has a lesser rate ${}^{O}R_i^{in}$. It is always the case that ${}^{O}R_i^{in} <= {}^{C}R_i^{in}$, otherwise our SLA would be violated by the consumer — she would be making requests at a higher rate than agreed and would have the right to reject them without paying a fine. Furthermore, we assume that for each cloud service provider p_j we know the:

- C_j^{out}: income received per request;
- P_j^{out}: penalty paid when a request is not serviced;
- ${}^{C}R_j^{out}$: contractually agreed/negotiated average rate of service requests; and
- ${}^{O}R_j^{out}$: its observed/assumed average request rate.

Our observed/assumed rate ${}^{O}R_j^{out}$ may be greater than ${}^{C}R_j^{out}$, as there is nothing forbidding the provider to serve requests faster than promised. If it is lesser than ${}^{C}R_j^{out}$, then the provider may end up paying a penalty, assuming that consumer requests are forwarded at the agreed rate ${}^{C}R_j^{out}$.

However, it is not possible to know the actual provider's rate of service processing or how likely it fails to serve a request. This would require knowing the length of its internal buffer, which is part of the provider's internal business information that we cannot know. We can however consider different scenarios for the number of lost service requests at the provider's side L_j^{out}. It should be stressed that all rates are average rates, so there are situations where they arrive too fast/slow or they can take more/less time to be served than the average. Thus, for an SLA to be agreed, we need to know:

- max consumer request(/min service) rate;
- penalty amount; and
- payment for the service.

7. SLA Monitoring

Monitoring an SLA is a critical process for the validation of the SLA guarantee terms. To initiate the monitoring process, a clear definition of the components of a Big Data analytics service is required, to know which operations or assets of the service need to be monitored. Once the service has been defined, the service user must be able to stipulate what functional and non-functional properties need to be associated with specific service operations.

In our approach, we use the EVEREST monitoring tool [42] to monitor the relevant service in a MBDAaaS computation, based on the defined guarantee terms of the SLA. Therefore, we assign the monitoring task to the EVEREST monitoring tool, but we also need to assign a task for the event capturing, so as the monitor receives all the events of the service and checks them against the monitoring rules. Capturing events is an important operation for the implementation of the monitoring process. Collecting the events at the right point of execution is crucial for the correct evaluation of the security properties that are monitored. Each property usually requires its own types of events that need to be captured and monitored. When the big data analytics service gets executed, a series of operations get executed and therefore multiple event captors might need to be installed. The events that are going to be captured are going to be either information with respect to the data that are processed itself or other metadata referring to the execution context of the operations. We decided to assign the task of the event capturing to the MBDAaaS platform for accuracy and flexibility reasons, since both the monitoring and the event capturing process should not be part of the actual service execution.

For the monitor to operate, a set of monitoring rules must be defined to decide upon possible violations or satisfactions. The definition of the monitoring rules, however, is intrinsically connected with the structure of the service. This is because the definition of the rules is dictated by the big data analytics operations that the service is comprised of. In our case, the generation of the monitoring rules and the installation of the event captors is an automated process that requires no intervention from the user. Every time a new instance of a service runs, it is the responsibility of the monitoring framework to find the relevant operations or data assets, generate the rules and based on the rules to enable the event capturing capabilities and then perform the monitoring process.

Table 2: Event Calculus fundamental axioms

Predicate	**Description**
Initially(f)	f is true at time 0
HoldsAt(f, t)	f is true at time t
Happens(e, t)	Event e occurs at time t
Initiates(e, f, t)	If event e occurs, then f is true after t
Terminates(e, f, t)	If event e occurs, then f is false after t

To implement an SLA monitoring solution, we need to first provide a set of semantics that allow the user to express the guarantee terms of the SLA. Our monitor is a run-time monitoring engine built in Java that can reason on events against rules that are based on Event Calculus (EC) [41]. Event calculus is a first-order temporal formal language that can be used to specify properties of dynamic systems, which change over time. Such properties are specified in terms of events and fluents. Events are what cause a change to the state of the system that is monitored; fluents are used to keep track of the state of a system, which are initiated and terminated by events. For instance, the Integrity rule, utilizes the fluents to check if prior to a read event, the monitor observed a write event.

Table 2 presents a list with all the available operations that can be used in the event calculus language, to express the changes in a system through the occurrence of events. An event is denoted by e, a fluent by f, and the time by t. In our approach, the monitoring rules are extracted from the guarantee terms defined in an SLA and then translated into the EC language of the monitor. This is done based on EC templates, with the user only defining the values of the parameters.

8. Evaluation

Following our reference scenario in Section 3.1, we define the security properties to be monitored and propose an evaluation of the negotiation process quality.

8.1. Settings

With the advent of the General Data Privacy Regulation (GDPR) in Europe, the problem of correctly managing sensitive data is exacerbated even more and involves all enterprises. Privacy and security requirements can be

specified in the declarative model defining requirements on data anonymization, confidentiality, and integrity. These requirements are then used by MBDAaaS to generate an executable deployment model of the Big Data processes, and to specify information on the target platform and its configuration. All declarative and deployment information is fed into the SLA specification and negotiation tool, to obtain the rules to be evaluated on the Big Data infrastructure to show its level of compliance to user expectations. In the following we consider two main GDPR requirements emerging from our reference scenario in Section 3.1:

1. *Data Integrity at rest* (data and platform assurance). GDPR (articles 24 and 25) mandate to safely store usersâĂŹ data, guaranteeing their integrity. Cryptographic techniques could be deployed to guarantee data integrity (e.g., hash techniques). An assurance manager should verify that data hashes are always consistent, and that the hashing components are behaving correctly. In our approach, we are able to verify this requirement by checking that all the data have not been compromised during the overall process.
2. *Availability.* GDPR (article 32) mandates the ability to ensure the ongoing availability of services, as well as the ability to restore the availability in order to be able to access personal data in a timely manner. Thus, we continuously monitor this requirement based on a threshold defined in the SLA and notify the stakeholders for any possible violation of it.

We note that, driven by the above declarative requirements, a deployment model is generated specifying the details of the integrated privacy services. These high-level requirements and information on privacy components are the basis of the specification of SLAs in Section 5 based on rules specified in Section 8.2.

8.2. Monitoring Rules

We formally define rules for properties data integrity and availability.

8.2.1. Data Integrity Monitoring Rule

Property data integrity examines whether the data read by an operation are the same data produced by a previous operation. To achieve this, we intercept every *read* and *write* event of the service operations. Every time an operation reads an input item an event is registered and logged for the

purpose of evaluating it against the defined monitoring rule. Similarly, every time an operation writes data at one of its output items, those events would also be registered and logged. In both cases, it is necessary to log the hash code of the data item that is processed before and after the processing. It is then possible for the monitor to compare values and evaluate whether the data items that an operation reads are the same data items that its previous operation has written (e.g., by comparing the associated hashes). If this principle does not hold true, then the monitor raises a data integrity violation. The event calculus formula that expresses the monitoring rule for monitoring data integrity is:

$$\textbf{Happens}(READ(op2(), hc), t) \rightarrow$$
$$\textbf{Happens}(WRITE(op1(), hc), [0, t])$$

The formula states that when happens an event *READ* of an operation *op2()* that reads data with a hash code *hc* at time *t*, then it should have happened another event *WRITE* of an operation *op1()* that wrote these data with the same hash code *hc*, at time $t'<t$. The formula describes the fact that when an operation reads a data item, at some earlier time, some other operation must have written the same data item.

8.2.2. Availability Monitoring Rule

Property availability reflects how much time has elapsed from the moment a request has been made until the time it returns its result, that is, how fast the result of the data processing becomes available to the end-user or any other system that might have requested it. To measure the elapsed time for a particular operation, we need to provide timestamps for the start and end points of execution for the methods of all RDDs that are produced during the computation. Thus, the monitoring rule for the availability is the following:

$$\textbf{Happens}(start(appId, rddId, partId), t) \rightarrow$$
$$\textbf{Happens}(end(appId, rddId, partId), t + d)$$

The above rule states that when a *start* event happens in a specific *appId*, *rddId* and *partId* at time *t*, then another *end* event must happen in the exact same *appId*, *rddId* and *partId* within time *t+d*. The value of *d*, that is, the elapsed time, is provided from the end user and is part of the SLA guarantee term. If the time period that it represents is greater than the value provided, then the monitoring rule is violated.

Listing 3: The consumer module

```
1  const double reqRate;
2  module Consumer1
3   [req1] (true) -> reqRate: true;
4  endmodule
```

Listing 4: The provider (server) module

```
1  #const s#
2  const double srvRate;
3  const int s=#s#;
4  module Provider1
5   [srv1] (true) -> srvRate: true;
6  endmodule
7  #for i=2:s#
8  module Provider#i#=Provider1 [srv1=srv#i#]
9  endmodule
10 #end#
```

8.3. Negotiation Process Evaluation

Based on the *Platform assurance* layer described in Section 4, information regarding the deployed components and their configuration can be used regarding the maximum/minimum request rate, which is required for the negotiation process. Reward and penalty amounts should be defined by the SLA modeler. Thus, to start a negotiating process for a new SLA, we first need to define the:

- boundaries of the SLA terms that can be agreed; and
- rewards — penalties and charges to be negotiated.

By defining this values, we can evaluate the SLA offer and facilitate the SLA negotiation process, by using the Prism tool [12, 46]. Prism model checker processes these values and produces figures of the possible behavior of the SLA, so that we can make a decision about accepting or rejecting. Since we need the request rate, we base our formal model on the (finite-buffer) M/M/s/K queue [47] that has an incoming rate of requests λ, s servers with a rate μ of servicing these requests each, and a *buffer* with capacity K to help with request/service spikes (since the rates are average ones). The resulting PRISM model [46] is a continuous-time Markov chain (CTMC), given our choice of M/M/s/K queues. Listing 3 shows the basic model for consumers and Listing 4 for cloud service providers. Each of these has a single transition, which fires with a specific rate (currently all consumers have the same rate and so do all providers). All consumers use the same module definition as consumer 1 (see Listing 3) renaming its action and rate — similarly for providers, *e.g.*, lines 7–10 of Listing 4.

Listing 5: The MBDAaaS platform module

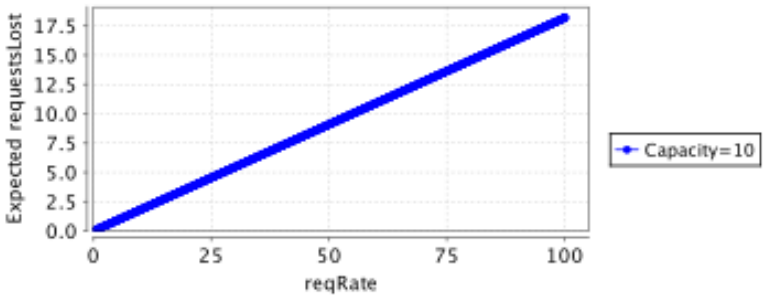
```
1  const int Capacity;
2  module Queue_M_M_s_K
3    waiting : [0..Capacity] init 0;
4    // Clients drive these actions
5    [req1] (waiting<Capacity) -> 1 : (waiting'=waiting+1);
6    [req1] (waiting=Capacity) -> 1 : true; // lost req
7    // Service providers drive these actions
8  #for i=1:s#
9    [srv#i#] (waiting>=#i#)
10     -> 1 : (waiting'=waiting-1);//serve if enough requests
11 #end#
12 endmodule
```

The module in Listing 5, representing the MBDAaaS platform, links consumers and providers together. The transitions of this module are synchronized with the same-named transitions of the other modules. This module defines their rate as 1.0, thus their global rate is the one declared in the other modules (rates of synchronized transitions in PRISM are multiplied together). Each time a consumer makes a request and there are free slots in the waiting queue, the MBDAaaS module accepts the request, increasing the number of waiting requests. If the waiting queue is full this module rejects the request. Furthermore, it permits one provider to serve a request when its queue is not empty and there are enough requests waiting to be served. It should be noted that the model is parametric on the number s of servers (see lines 8–11), and this number can be extracted from the deployment model. An example of the effect of the buffer length is shown in Figure 5(a) and Figure 5(b) for the cases where the incoming and outgoing rates are equal and range from 0.1 to 100.1, respectively. When the buffer capacity is set to 10 we lose $\approx 18\%$ of the consumer requests, while only $\approx 2\%$ when set to 100, allowing it to smooth out more spikes in demand.

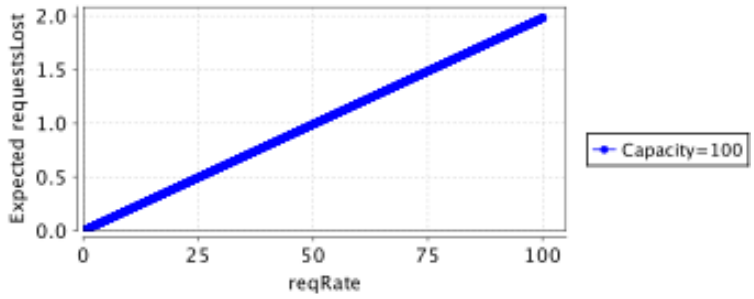
8.4. SLA *deployment*

By providing information regarding the deployment of the components and their configuration, as well as information regarding penalties or rewards that may apply for the specified requirements, SLAs can be generated. These agreements hold all information regarding security requirements for the involved services or service operations. Afterwards, the SLA negotiation process can be performed, which evaluates the proposed SLAs, prior to them being signed.

Based on our reference scenario, a service consumer (*e.g.*, SAP) using our MBDAaaS platform can generate an SLA and trigger the SLA negotiation process. By providing all requirements needed to generate the three models



(a)



(b)

Figure 5: (a) Expected Lost Request (Capacity=10), (b) Expected Lost Request (Capacity=100)

of our methodology, an SLA offer can be produced, according to the process described in section 5. This offer is then translated into a PRISM model, to be evaluated by the PRISM tool, as presented in section 6.2. When the Prism tool validates the model, the results are translated into Jess rules (see 6) and are sent to the Jess rule engine. The Jess rule engine checks the values of the generated rules and tries to match them with existing values from a service provider(s), who can provide the services required by the service consumer. If the action from the Jess rule engine is *reject*, the offer is rejected and no SLA is created. If the action is *accept*, the offer is accepted and a new SLA is created. If the action is *set*, a counter-offer with new values is provided, which should be checked again, by restarting the whole process.

When an SLA is signed, a run-time monitor can start monitoring the specific services and assure that the relevant security requirements defined in the SLAs hold throughout a specified period of time (*e.g.*, until the expiration

time). To this aim, the SLA terms are translated into the EC language that the monitor understands. The monitor is then fed with translated terms and continuously returns the monitoring results.

9. Conclusions

We provided a solution to *Big Data Assurance* based on SLAs, which integrates with a MBDAaaS solution. Our approach implements a SLA specification and negotiation process at all three layers of assurance, which addresses Big Data peculiarities. Furthermore, we have also integrated the SLA management framework of the proposed MBDAaaS platform with a run-time monitoring, to provide a complete *Big Data assurance* process based on continuous monitoring.

Acknowledgements

This work has received funding from European Union's Horizon 2020 research and innovation programme under grant agreement No H2020-688797 and grant agreement no. 830927, and by the programme "Piano sostegno alla ricerca 2018" funded by the Università degli Studi di Milano.

Bibliography

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

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

[3] T. H. D. Patil, Data Scientist: The Sexiest Job of the 21st Century, harvard Business Review, https://hbr.org/2012/10/data-scientist-the-sexiest-job-of-the-21st-century (October 2012).

[4] W. Markow, S. Braganza, B. Taska, S. M. Miller, D. Hughes, The Quant Crunch – How the Demand for Data Science Skills is Disrupting the Job Market, https://www.ibm.com/downloads/cas/3RL3VXGA (2017).

[5] C. Ardagna, V. Bellandi, M. Bezzi, P. Ceravolo, E. Damiani, C. Hebert, Model-based big data analytics-as-a-service: Take big data to the next level, IEEE Transactions on Services Computing (TSC) (2018).

[6] E. R. Sparks, S. Venkataraman, T. Kaftan, M. J. Franklin, B. Recht, Keystoneml: Optimizing pipelines for large-scale advanced analytics, in: Proc. of IEEE International Conference on Data Engineering (ICDE 2017), San Diego, CA, USA, 2017.

[7] C. Ardagna, V. Bellandi, P. Ceravolo, E. Damiani, R. Finazzo, A methodology for cross-platform, event-driven big data analytics-as-a-service, in: Proc. of 5th International Workshop on Methodologies to Improve Managing Big Data projects, Los Angeles, CA, USA, 2019.

[8] Improvado, All Your Marketing Data in One Place, https://improvado.io/.

[9] Databricks, Unified Analytics, https://databricks.com/.

[10] C. Ardagna, R. Asal, E. Damiani, Q. Vu, From security to assurance in the cloud: A survey, ACM Computing Surveys 48 (1) (2015) 2:1–2:50.

[11] C. Ardagna, E. Damiani, M. Krotsiani, C. Kloukinas, G. Spanoudakis, Big data assurance evaluation: An sla-based approach, in: Proc. of the IEEE International Conference on Services Computing (SCC 2018), San Francisco, CA, USA, 2018.

[12] M. Z. Kwiatkowska, G. Norman, D. Parker, PRISM 4.0: Verification of probabilistic real-time systems, in: Proc. of the 23rd International Conference on Computer Aided Verification (CAV 2011), Snowbird, UT, USA, 2011.

[13] IATAC and DACS, Software Security Assurance: State of the Art Report (SOAR), https://tinyurl.com/y9y9zvkv (July 2007).

[14] M. Anisetti, C. Ardagna, E. Damiani, F. Saonara, A test-based security certification scheme for web services, ACM Transactions on the Web 7 (2) (2013) 1–41.

[15] M. Krotsiani, G. Spanoudakis, K. Mahbub, Incremental certification of cloud services, in: Proc. of the 7th International Conference on Emerging

Security Information, Systems and Technologies (SECURWARE 2013), Barcelona, Spain, 2013.

[16] M. Anisetti, C. Ardagna, E. Damiani, F. Gaudenzi, A semi-automatic and trustworthy scheme for continuous cloud service certification, IEEE Transactions on Services Computing (TSC) (2017).

[17] M. Anisetti, C. Ardagna, E. Damiani, A. Mana, G. Spanoudakis, Towards transparent and trustworthy cloud, IEEE Cloud Computing Magazine 4 (3) (2017) 40–48.

[18] S. Lins, S. Schneider, A. Sunyaev, Trust is good, control is better: Creating secure clouds by continuous auditing, IEEE Transactions on Cloud Computing PP (99) (2016) 1–1. doi:10.1109/TCC.2016.2522411.

[19] O. A. Wahab, J. Bentahar, H. Otrok, A. Mourad, Towards trustworthy multi-cloud services communities: A trust-based hedonic coalitional game, IEEE Transactions on Services Computing (TSC) 11 (1) (2016) 184–201.

[20] A. Sunyaev, S. Schneider, Cloud services certification, Communications of the ACM 56 (2) (2013) 33–36.

[21] P. Stephanow, G. Srivastava, J. Schütte, Test-based cloud service certification of opportunistic providers, in: Proc. of the 9th IEEE International Conference on Cloud Computing (CLOUD 2016), San Francisco, CA, USA, 2016.

[22] A. Verma, L. Cherkasova, R. H. Campbell, Aria: Automatic resource inference and allocation for mapreduce environments, in: Proc. of the 8th ACM International Conference on Autonomic Computing (ICAC 2011), Karlsruhe, Germany, 2011.

[23] E. Hwang, K. H. Kim, Minimizing cost of virtual machines for deadline-constrained mapreduce applications in the cloud, in: Proc. of the 2012 ACM/IEEE 13th International Conference on Grid Computing (GRID 2012), Beijing, China, 2012.

[24] Y. Wang, W. Shi, On scheduling algorithms for mapreduce jobs in heterogeneous clouds with budget constraints, in: Proc. of the 17th International Conference on Principles of Distributed Systems (OPODIS 2013), Hong Kong, China, 2013.

[25] P. Lama, X. Zhou, Aroma: Automated resource allocation and configuration of mapreduce environment in the cloud, in: Proc. of the 9th International Conference on Autonomic Computing (ICAC 2012), San Jose, CA, USA, 2012.

[26] S. V. Hrushikesha Mohanty (Ed.), Big Data Service Agreement, Springer, 2015.

[27] A. Andrieux, K. Czajkowski, K. Keahey, A. Dan, K. Keahey, H. Ludwig, J. Pruyne, J. Rofrano, S. Tuecke, M. Xu, Web services agreement specification (WS-Agreement), in: Global Grid Forum GRAAP-WG, Honolulu, HI, USA, 2004.

[28] H. Ludwig, A. Keller, A. Dan, R. P. King, R. Franck, Web service level agreement (WSLA) language specification, Tech. rep. (2002). doi:10.1109/WECWIS.2002.1021238.

[29] S. Nepal, J. Zic, S. Chen, WSLA+: Web service level agreement language for collaborations, in: Proc. of IEEE Internation Conference on Services Computing (SCC 2008), Honolulu, HI, USA, 2008.

[30] K. T. Kearney, F. Torelli, C. Kotsokalis, SLA*: An abstract syntax for service level agreements, in: Proc. of 11th ACM/IEEE International Conference on Grid Computing (Grid 2010), Brussels, Belgium, 2010.

[31] D. D. Lamanna, J. Skene, W. Emmerich, SLAng: A language for defining service level agreements, in: Proc. of the IEEE Computer Society Workshop on Future Trends of Distributed Computing Systems, Tunis, Tunisia, 2003.

[32] Y. Kouki, F. A. D. Oliveira, S. Dupont, T. Ledoux, A language support for cloud elasticity management, in: Proc. of the 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid 2014), Chicago, IL, USA, 2014.

[33] R. B. Uriarte, F. Tiezzi, R. De Nicola, SLAC: A formal service-level-agreement language for cloud computing, in: Proc. of the IEEE/ACM International Conference on Utility and Cloud Computing (UCC 2014), London, UK, 2014.

[34] S. Madden, From databases to big data, IEEE Internet Computing 16 (3) (2012) 4–6.

[35] 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, in: Proc. of the International Workshop on Semantic Web Services and Web Process Composition (SWSWPC 2004), San Diego, CA, USA, 2004.

[36] J. Vaidya, B. Shafiq, A. Basu, Y. Hong, Differentially private naive bayes classification, in: Proc. of the 2013 IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT), Atlanta, GA, USA, 2013.

[37] Y. Demchenko, P. Membrey, P. Grosso, C. de Laat, Addressing big data issues in scientific data infrastructure, in: Proc. of the 1st International Symposium on Big Data and Data Analytics in Collaboration (BDDAC 2013), San Diego, CA, USA, 2013.

[38] N. Karten, How to Establish Service Level Agreements, 2003, http://www.nkarten.com/handbook.pdf.

[39] G. Rosu, T. Serbanuta, An overview of the K semantic framework, J. Log. Algebr. Program. 79 (6) (2010) 397–434.

[40] CUMULUS, D3.2 - core certification mechanisms v2, Tech. Rep. D3.2, CUMULUS EU project (2013).

[41] M. Shanahan, The event calculus explained, in: Artificial Intelligence Today, 1999, pp. 409–430.

[42] G. Spanoudakis, C. Kloukinas, K. Mahbub, The SERENITY runtime monitoring framework, in: Security and Dependability for Ambient Intelligence, 2009, pp. 213–237.

[43] A. Köppel, D. Böning, S. Abeck, How to support the negotiation of service level agreements (SLAs) for your client/server application, in: Proc. of the World Multiconference on Systemics, Cybernetics and Informatics (SCI 1999), Orlando, FL, USA, 1999.

[44] K. Mahbub, G. Spanoudakis, Proactive SLA negotiation for service based systems: Initial implementation and evaluation experience, in: Proc. of the IEEE International Conference on Services Computing (SCC 2011), Washington, DC, USA, 2011.

[45] E. F. Hill, Jess in Action: Java Rule-Based Systems, Manning Publications Co., Greenwich, CT, USA, 2003.

[46] G. Norman, D. Parker, J. Sproston, Model checking for probabilistic timed automata, Formal Methods in System Design 43 (2) (2013) 164–190.

[47] L. Kleinrock, Theory, Volume 1, Queueing Systems, Wiley-Interscience, 1975.