IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received February 4, 2021, accepted February 16, 2021, date of publication March 17, 2021, date of current version March 25, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3066443

A Flexible Billing Life Cycle for Cloud Services Using Augmented Customer Agreements

JOSÉ MARÍA GARCÍA[1,2], OCTAVIO MARTÍN-DÍAZ[2], PABLO FERNANDEZ[1,2], CARLOS MÜLLER[2], AND ANTONIO RUIZ-CORTÉS[1,2], (Member, IEEE)

[1]Smart Computer Systems Research and Engineering Laboratory (SCORE), Universidad de Sevilla, 41012 Seville, Spain
[2]Research Institute of Informatics Engineering (I3US), Universidad de Sevilla, 41012 Seville, Spain

Corresponding author: José María García (josemgarcia@us.es)

This work was supported in part by the European Commission (FEDER) and the Spanish Government through Project HORATIO under Grant RTI2018-101204-B-C21, and in part by the Andalusian Administration through Project APOLO under Grant US-1264651 and the Project EKIPMENT-PLUS under Grant P18-FR-2895.

ABSTRACT Cloud computing constant evolution requires dynamic adjustments to service pricing and billing terms, considering provider infrastructures, customer requirements, and discount policies, among others. In this context, Customer Agreements (CA) are used to regulate the service provision including, among other information, the agreed service level in SLAs, pricing, and billing terms. Although many existing proposals and industrial tools support the definition of pricing and billing of cloud services, there is a lack in terms of customisation and automated monitoring tools integrated with the billing process. In this article, we provide a flexible billing proposal supporting CA that considers not only SLA terms but also customisable pricing and billing terms, possibly including compensations (i.e. discounts or overcharges) that apply when specified conditions are met. In addition, we developed an automated monitoring and analysis tool that we validate in a real-world industrial scenario. Moreover, we analyse and compare more than 50 existing industrial tools with our proposal, highlighting the advantages of its rule-based approach.

INDEX TERMS Billing, cloud services, customer agreements, monitoring, pricing, service level agreements.

I. INTRODUCTION

Within the highly dynamic and evolving cloud computing market, a proper service pricing and billing life cycle definition is crucial for providers and consumers alike. On the one hand, providers need to customise their service offerings, taking into account several aspects such as their own infrastructure, the target market, incurred costs, discount policies, purchasing options, and the actual billing process life cycle, which can vary from fixed billing during a period of time to a real-time, fluctuating price based on demand-and-supply chains [1]–[3]. On the other hand, customers face an ever growing number of service offerings to choose from, where they need to analyse which are the optimal options with respect to their requirements and budget, as well as carefully monitor that the billing conforms to the Customer Agreement (CA) and especially to its associated Service Level Agreement (SLA) [4], [5], i.e. services are correctly billed according to the consumed resources, the service level actually offered, the contractual price, and applicable discounts, amongst other variables.

In this context, CAs usually consist of a textual document that is predefined by providers, and which is common to all their consumers. Mostly, their associated SLAs specify a number of service level objectives (SLO) on known metrics such as latency or availability, together with one or more compensations to be applied if not fulfilled [6]. In absence of automated tools, consumers are solely responsible for checking any SLA violations that may occur in order to claim for penalties to be applied in the following billing cycles.

Nevertheless, CAs in general and SLAs in particular can play a more relevant role in cloud infrastructure governance, enabling its automatic monitoring, dynamic adaptation to comply with the SLOs, and the application of compensations during the billing process. In order to do so, agreement documents should be augmented with additional information and features, such as higher flexibility and expressiveness on the metrics or SLOs, pricing models to be applied at service provision time, billing customisation, and compensations containing discount rules based on such pricing models, to be applied at billing time.

The associate editor coordinating the review of this manuscript and approving it for publication was Peng-Yong Kong.

In this article, we propose a flexible service billing process governed by CAs and their associated SLAs, so that providers can customise their billing life cycle with respect to the pricing model, discounts, and the metrics that are relevant to the process. The main contributions of our work can be summarised as follows:

1) We provide **a model to specify pricing and billing information** augmenting CA documents. Thus, we leverage billing life cycle definition as a first-class citizen in CAs and discuss how SLAs influence the activities carried out during the billing process.
2) We propose **an automatic billing generation based on rules** derived from our pricing and billing model. Our solution enables the dynamic optimisation of revenues according to actual use of infrastructures and services, gathered from service monitoring, while benefiting consumers with more accurate and possibly discounted bills.
3) We present **an analysis of automated billing systems and a prototype tool** that allows providers to customise the life cycle and generate bills according to the CAs they offer. We apply the developed tool in an industrial use case, evaluating its performance under different workloads, and comparing it with the 53 industrial tools we analysed.

The rest of the article is structured as follows. First, in Sec. II we present the real-world scenario that motivated this work and we identify the faced challenges. Then, Sec. III presents our proposal to dynamically generate bills with respect to monitored service level and discount rules. We validate our solution in Sec. IV, applying it to the motivating scenario, evaluating its performance, and also comparing some tools which can be used to support billing.[1] We discuss in Sec. V the related work regarding cloud pricing models and billing mechanisms. Finally, Sec. VI concludes the article, discussing our results and identifying future work.

II. MOTIVATION AND CHALLENGES

The contributions presented in this work are motivated in the context of a real scenario. In the following, we describe this scenario, from which we derive the actual challenges related to the cloud services billing life cycle.

A. MOTIVATING SCENARIO

Our scenario consists on a distributed provision of Software as a Service (SaaS) for educational platforms in the public administration of Andalusia (the biggest region of Spain in terms of size and population), which have to be dynamically deployed and scaled up to support online teaching needs. In this context, the Educational Administration (EA) contracts a SaaS provider (SP) to deliver the platforms for a network of over 2,000 schools distributed geographically across the region; each of those schools represent a community of end-users (EU) for a given provided educational platform. Specifically, the SP has to set and maintain a flexible infrastructure that can scale to dynamically deploy the requested instances of educational platforms (based on the Moodle Framework) by the EA. Furthermore, privacy regulations oblige the chosen provider to satisfy all the privacy and data protection needs, initially resulting in a requirement to host those platforms in a private cloud infrastructure deployed in the data centres owned by EA.

In order to address these requirements, a container-based solution was implemented; this approach provides a flexible deployment and operation model that allows the reconfiguration and evolution required in the scenario. Specifically, Fig. 1 shows the architecture of the platform based on the contaniner orchestrator *Kubernetes* where all the components are deployed. In this environment, the platforms manager operates (from SP) the system with a provisioning tool that generates the educational platforms instances on demand, based on the requests from the EA, setting up the appropriate persistency layer by means of a clustered database with the appropriate configuration registry and data backups. From a consumer standpoint, each school represents a set of EU that access their dedicated instance by means of the Kubernetes proxy. All those components are orchestrated by the k-master core module that is responsible to integrate and monitor all the containers.

In this context, the initial scenario was a fixed reserved physical server owned and operated by the SP but installed in the EA data centers. This resulted in a recurring bill depending on the number of platforms deployed without taking into account the actual usage of each platform. Unfortunately, this initial model proved to be unsatisfactory for both the EA and the SP. On the one hand, from the EA perspective, even though there was a dynamic billing depending on the number of platforms used on every day, there was also a minimum base cost derived from the maintenance and operation of the fixed reserved infrastructure. On the other hand, from the SP perspective, there was a continuous under-usage of the infrastructure that could not be used for other customers in order to generate new business possibilities, and increase its revenue.

To overcome these limitations, both the EA and the SP acknowledged to explore an alternative pricing model to migrate the reserved physical server to a data center owned by the SP as a private cloud for its customers. By using this shared infrastructure model, it is still feasible to reserve parts of the infrastructure, but during periods when usage loads by the administration are low (e.g. at nights or on vacations) the provider can allocate more computing resources to other customers with the subsequent gain, while the administration gets a discount for the low usage load. In this scenario, a CA was settled defining a set of rewards for the administration in case the reserved resources (in terms of computing, memory or disk) were not used during a certain period.

[1] A landing page with supplementary material including our prototype and the comprehensive comparative analysis is available at https://isa-group.github.io/2020-10-billing-lifecycle/

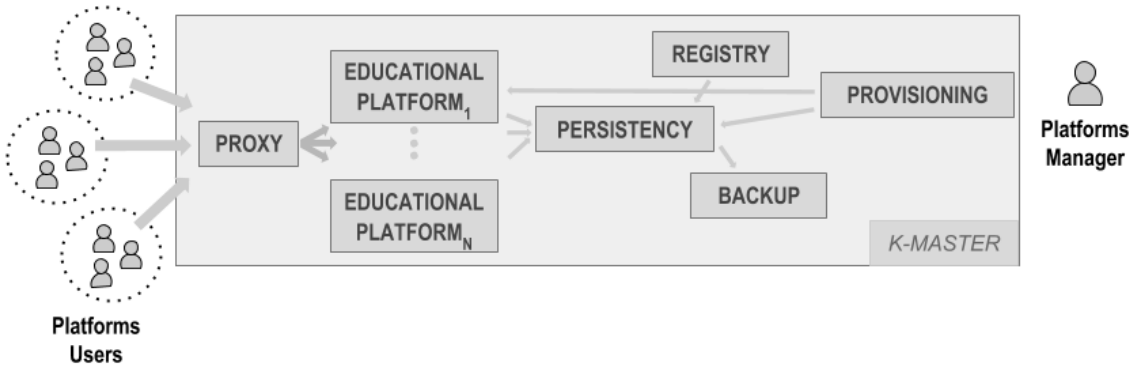


FIGURE 1. Motivating scenario.

B. CHALLENGES TOWARDS CUSTOMISATION OF THE BILLING LIFE CYCLE

The dynamic scenario previously described poses a series of requirements with respect to scalability, adaptability, cost optimisation, and privacy issues that impose significant challenges to achieve a truly flexible and customisable billing life cycle:

1) Specification of all billing-related information within the same CA together with the relevant service information. Since billing rules are based on service metrics, which in turn are usually defined in relevant CAs, they should be considered as a first-class citizen. Thus, CAs augmented with billing specifications should drive the billing life cycle of associated services, including contracting, monitoring, charges and billing generation. By addressing this challenge, solutions will enable easier *adaptation* and *customisation* of service offerings.
2) Highly-expressive billing rules, supporting the definition of not only pricing terms depending on different metrics, but also discounts and compensations. This expressiveness will facilitate *cost optimisations* for both consumers and providers.
3) Automated analysis tools, supporting metrics monitoring, billing generation, and simulation. Billing support systems should provide analytical platforms to automatically compute bills according to metrics gathered from the cloud infrastructure. Additionally, these tools should take *scalability* and *privacy* issues into account.

III. CUSTOMISABLE BILLING GENERATION PROCESS

In order to allow the level of customisation required for the dynamic billing scenario described before addressing the identified challenges, we propose a billing generation framework based on CAs that are augmented with SLAs, pricing and billing rules, in addition to the service terms. In the following sections, we exemplify our proposal using the results from an industrial research project named POETISA, which was conducted to provide a solution to the challenges identified in Sec. II-B.

A. CHARACTERISING THE BILLING LIFE CYCLE

Figure 2 shows a representation of a prototypical billing life cycle using a BPMN diagram. The process begins with the *service provision*, which establishes the relationship between provider and customer, enacting a CA which usually contains the pricing information, billing rules that defines billing periods and conditions to generate the actual bills, as well as various terms that specify the provided service. This information is considered within the proper billing cycle, as delimited by the billing rules included in the CA, where the *monitoring* and the *billing* subprocesses are performed iteratively for the provided service, until the service is *decommissioned*. All these different tasks comprising the billing life cycle may vary from one provider to another, offering custom made alternatives depending on their business capabilities and the market characteristics. These tasks and some common alternatives are further discussed in the following.

1) SERVICE PROVISION

Whenever a customer wants to make use of a service, they must choose between offerings that provide different configurations regarding the service capabilities as well as the billing life cycle instantiation. The latter results in varying schemata that define billing periods, charge dates, and discounts based on the options finally chosen. Among the most common options, not necessarily exclusive, are the following:

- *Pay-on-demand.* The most simple and usual option, where customers pay for services which they have actually used during a billing period. Customers usually just need to register on the service, so that any later usage implies implicitly the CA acceptance, which is usually predefined, and the subsequent billing. However, resources can be unavailable because of outages or limitations, such as too many customers accessing the service at the same time.
- *Reservation.* Customers request the exclusive use of a resource for a period, such as a year, in order to avoid problems of unavailability. Reservations make the customers eligible for discounts. Amazon was one of first providers to offer reserved instances to customers.
- *Auction.* Providers offer a dynamic, usually cheaper price for unused resources, such as reservations not currently being active, to increase their revenue. Thus, customers bid for their usage, with the risk that such

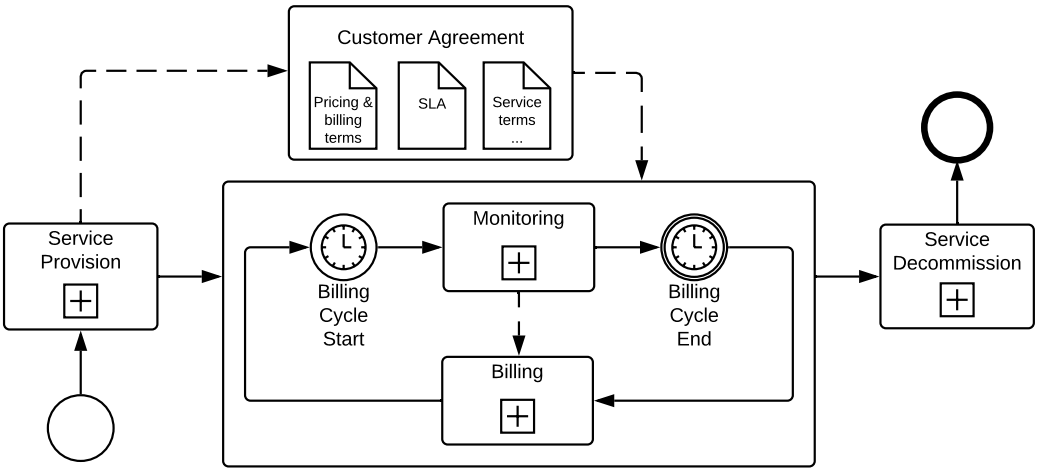


FIGURE 2. Process diagram representing a prototypical billing life cycle.

usage can be interrupted in case the actual customers who reserved them need to use those resources. As an example, Amazon offers the so-called *spot* instances as a kind of auction to its customers.

- *Subscriptions.* Customers request a period of flat-rate usage of resources, such as a year, or a personalised plan. Subscriptions usually have the largest discounts. Rackspace was one of first providers to offer pre-payment options to customers.

2) MONITORING AND BILLING CYCLE

Once the CA has been enacted after provisioning the service, the actual billing cycle starts according to the conditions stated in the CA. During a billing cycle, providers monitor the service usage to gather relevant metrics so that they can afterwards compute the bills with relevant data. Both monitoring methods and data should be accessible to customers, in order to allow them to validate the billing process and claim any miscalculation due to SLA violations not considered by the provider. Unfortunately, it is not always that way [7].

After the billing cycle ends, the billing subprocess depicted in Fig. 2 takes the monitoring data as well as the pricing and billing terms in order to generate bills and send due charges to the customer account. Depending on the billing life cycle chosen at service provision time, we can classify charge models as follows:

- *Prepayment.* Customers pay a single, preliminary charge, including larger discounts, in case of *subscriptions*, such as Rackspace options.
- *Upfront.* Customers pay a preliminary payment in case of *reservations*, like with Amazon reserved instances. Later, ordinary usage is charged as well, but with some discounts.
- *Ordinary.* Customers pay periodically for ordinary usage, usually monthly, including discounts in case of reservation or auction.

3) SERVICE DECOMMISSION

Finally, the billing life cycle comes to an end depending on the conditions and terms agreed upon in the CA. Most common conditions that trigger the service decommission subprocess are the following:

- *Expiration.* CAs may include an expiration date in their service terms, as when subscriptions are in place.
- *Disuse.* Customers simply stop using the resources. On pay-on-demand schemata, if resources are not used, then there are no charges, since the CA does not apply.
- *Cancellation.* CAs may include explicit termination conditions or other limitations, such as a maximum number of API invocations during a period (as Amazon API Gateway), or under a termination fee (as Rackspace's Master Services Agreement).

4) CONTEXT OF OUR SOLUTION

As we discussed in Sec. II-B, in order to address the identified challenges we need to provide a high degree of flexibility to the whole billing life cycle presented above. Focusing on the monitoring and billing subprocesses, which are fundamental to our proposal, both providers and customers have different perspectives and expectations towards dynamic management and customisation of the billing life cycle:

- *Billing generation.* On the one hand, providers clearly need to bill their customers according to the actual usage of the relevant services. In order to do so, both accessing the monitoring data and appropriately applying billing rules (pricing and discounts) are mandatory features.
- *Billing validation.* On the other hand, customers should have the capacity for using the same billing rules to check that the billing carried out by the provider conforms to their agreement, thus avoiding discrepancies between them that could lead to legal claims.
- *Billing estimation.* Apart from validation, customers may also require the simulation of subsequent billing periods [8]. This opens opportunities for comparing with other providers for strategical and/or tactical purposes.

Nevertheless, there are situations where a business entity may act as service consumer and provider at the same time, which leads to intertwined needs related to billing generation,

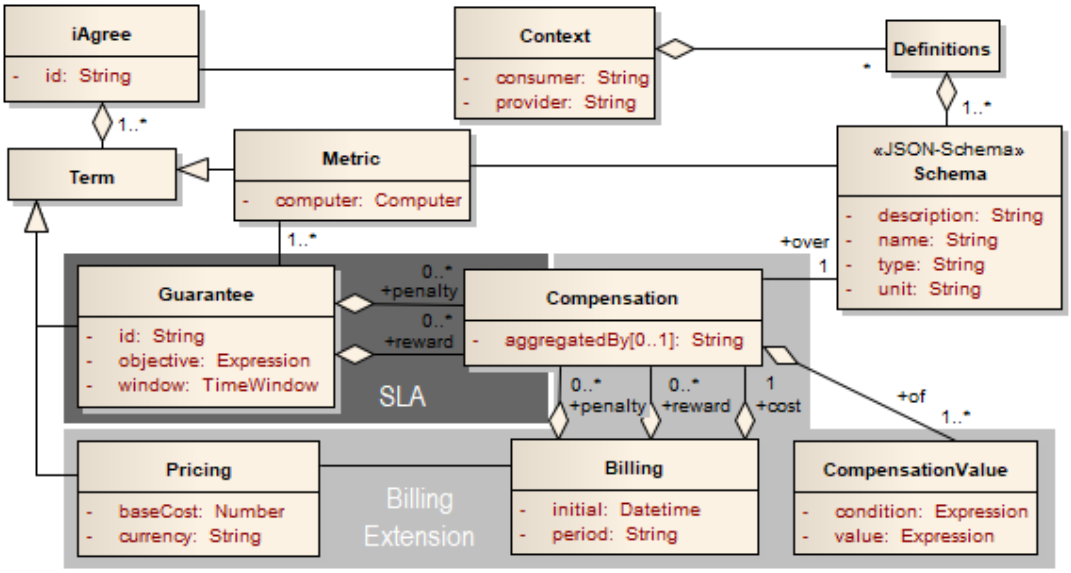


FIGURE 3. UML model excerpt of iAgree elements including our proposed Billing extension.

validation, and estimation. In the following, we focus our contribution on the billing generation scenario, proposing a solution that addresses the previously identified challenges based on dynamic pricing models specified alongside SLAs, serving as the foundations to customise the billing generation process.

B. PRICING MODEL SPECIFICATION

Our proposed pricing model is included within `iAgree`,[2] an agreement model that is partially depicted in Fig. 3. iAgree faces the first research challenge we identified in Sec. II-B since it supports modelling CAs including both SLA terms (c.f. dark grey area) and billing terms by means of our proposed extension (c.f. light grey area). To further illustrate this extension, we also provide in Fig. 4 an excerpt of the CA which corresponds to the motivating scenario exposed in Sec. II-A. Line references in the text below refer to such excerpt.

The iAgree model is basically comprised of a set of *Terms* and some *Context* information, such as the parties involved in the agreement and a set of JSON-Schema *Definitions* to be used in metrics, objectives, and compensations. As an example, the `BillingDiscount` schema at lines 8-10 is used in the reward describing a billing discount at line 28.

In this model, *Pricing* appears as a first-class term, just as *Metrics* or *Guarantees*. On the one hand, a Metric term includes its schema, and a computer URL that references a RESTful API that is required to monitor its current values. As an example, the URL at line 40 provides the average memory consumption (AMC) of the cloud infrastructure. On the other hand, a Guarantee term specifies the service level objective to be fulfilled during a time window, as the `Uptime` $\geq$ `99.00` condition at line 45. Optionally, as presented in [6] a compensable guarantee may penalise or reward the under or over-fulfilling of the service level objective, respectively.

Focusing in our proposed extension, a *Pricing* term specifies the current currency and a base cost. Starting from this base cost, additional quantities can be added or subtracted for a specific *billing* period by considering a set of *compensation* values over the cost schema in a similar way as proposed compensations for guarantees. Note that a pricing compensation refers to either an overcharge or a discount instead of penalties and rewards on service level. As an example, the Daily Platform Cost (DPC) is established at line 19 as the base cost per educational platform with a value of `0.78`. Since our proposed model supports highly-expressive cost specifications by means of rules, the final cost at lines 21 and 24 not only considers the DPC, but also the Number of Moodle platforms (NM) and the Number of Days of the Month (NDM), which are additional metrics. These final cost formulas apply if conditions on NM in lines 22 and 25 hold, respectively.

Note that the billing rule specified in line 24 denotes that the billing process in this example follows a *reservation* schema, because the final cost is not affected up to 150 NMs. If the provider wanted to offer a *pay-on-demand* option instead, it would just specify a higher base cost (e.g. 0.9), removing the conditional terms with respect to the NMs. Furthermore, the final cost in the scenario can be also modified by a monthly billing discount of 10% if the AMC metric is less than 30 Gb, as defined in the reward in line 28.

C. AUTOMATIC BILLING GENERATION

From the pricing and billing terms specified using the format described above, we derive a set of rules that are evaluated using a rule engine [9]. Thus, as shown in Fig. 5, our proposal first analyses the iAgree specification in order to extract the metrics information by using each computer URLs

[2]http://iagree.specs.governify.io/Specification/

```
id: POETISA_CUSTOMER_AGREEMENT:
context:
  provider: https://www.isa.us.es/
  consumer: http://www.juntadeandalucia.es/
  ...
  definitions:
    schemas:
      BillingDiscount:
        description: Percent to reduce next monthly bill
        type: double; unit: '%'
terms:
    pricing:
        currency: EUR
        billing:
            period: monthly
            initial: '2018-04-28T10:35:36.000Z'
            cost:
                over: MonthlyCost
                    baseCost: DPC: 0.78
                of:
                    value: DPC * NM * NDM
                    condition: NM ≥ 150
                of:
                    value: DPC * 150 * NDM
                    condition: NM < 150

            rewards: //discount compensation
                over: BillingDiscount
                aggregatedBy: sum
                of:
                    value: 10
                    condition: AMC < 30
            ...
    metrics:
    AMC:
      schema:
        description: Average memory consumption
        type: double; unit: 'GB'
        minimum: 0; maximum: 100
      computer: http://127.0.0.1/api/v2/AMC
    ...
    guarantees:
      id: G_Uptime
        of:
            objective: Uptime ≥ 99.00
            window: ...
```

FIGURE 4. Excerpt of POETISA Customer Agreement in *iAgree*.

declared (as in line 40 of Fig. 4). We query those computers components specifying the relevant time window to generate the corresponding bill. The values obtained are injected into the rule engine's working memory as facts. For instance, if the AMC obtained for a given time window were 20 GB, our solution would introduce the fact $AMC = 20$ into the working memory of the rule engine.

Together with the metrics information, our proposal transforms the iAgree conditions and values into a rule with the format `condition` → `action`. For instance, the discount compensation specified in lines 28-32 of Fig. 4 corresponds to the following rule:

$$AMC < 30 \rightarrow BillingDiscount \mathrel{+}= 10$$

Note that line 29 specifies the aggregation operator in case several rules are applied over the same billing parameter (i.e. `BillingDiscount`). These rules are injected into the rule engine's production memory.

Then, after transforming the pricing model specification into rules and obtaining the metrics values as facts, we make use of a rule engine (we use json-rules-engine[3] in the prototype described in Sec. IV-A), so that we can compute discounts and final price according to the iAgree specification.

[3]https://www.npmjs.com/package/json-rules-engine

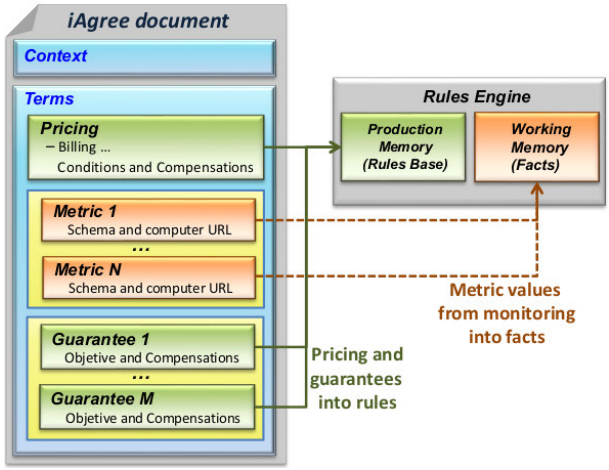


FIGURE 5. Transformation from iAgree to rules.

For instance, considering the previously described rule and the fact containing the AMC metric value, the rule engine would fire that rule, since the condition holds, modifying its working memory to indicate a 10% increment for the billing discount parameter. After the rule engine finishes processing the rules, our solution queries the working memory to retrieve the corresponding billing parameters to generate the final bill.

By using this approach, our solution addresses the second challenge identified in Sec. II-B, taking advantage of the expressiveness of rule engines to describe complex conditions and generate bills using arbitrary formulas for computing discounts and service costs, in terms of the `action` part of the rules obtained from the analysis and transformation of the iAgree document.

IV. TOOLING SUPPORT AND VALIDATION

In order to evaluate the suitability of our conceptual solution, we developed a software prototype that was deployed in the context of the real motivational scenario described in Sec. II, addressing the third challenge discussed in Sec. II-B. Based on this deployment, a validation process was carried out to gather metrics from the live platform and perform a billing analysis for the provider during the course of an industrial project called POETISA.

In this section, we detail the structure of the implemented prototype, outline the POETISA use case validation, and analyse its performance under different conditions. Finally, a review of the current industrial tooling support is offered, not only providing an analysis of the features supported by most relevant commercial tools, but also pointing out the strengths and weaknesses of our proposed tooling.

A. PROTOTYPE ARCHITECTURE

From an architectural point of view, Fig. 6 depicts the different elements that conform the validation scenario; specifically, this architecture includes an ecosystem of components (at the bottom part of the figure) that are integrated with the preexisting provisioning infrastructure (at the top part of the figure) for educational platforms. The overall system is

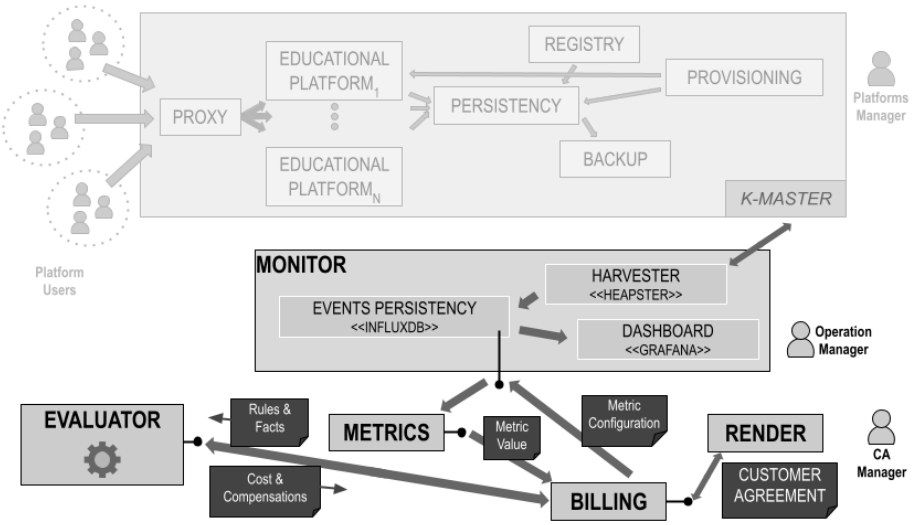


FIGURE 6. Prototype architecture.

deployed into a Kubernetes platform that provides a management infrastructure for architectures based on containers to support reliability and availability. Specifically, the prototype is deployed in a single-node Kubernetes cluster (k-master) that could be extended with additional nodes (k-minions) to extend the computing capabilities; in this context, the k-master node provides an extensive API to actuate and monitor all the artefacts deployed in the infrastructure.

From a global standpoint, in Fig. 6 we can see how the *Customer Agreements* are managed and updated in the system by the *CA Manager* using the **Render** component that triggers (on behalf of the manager) the different billing computations showing back its results; the billing computation itself is orchestrated by the **Billing** component that gather the appropriate *Metrics Values* from the **Metric** component and generates the set of *Rules* and *Facts* that should be resolved in the **Evaluator** component that calls a rule engine. It is important to highlight, that this general process described is based on background parallel monitoring activity developed by the **Monitor** component that is configured by the specific *Metric configurations* sent from the **Billing** component in order to harvest the actual monitoring data using the *Heapster* framework and collect them in the time series database *InfluxDB*, which the **Metric** component can efficiently query later on.

Specifically, the prototype consists of the following elements:

- **Monitor** component integrates *Heapster* (responsible for gathering events monitored from the cluster), *InfluxDB* (a persistence store with query capabilities specialised in time-series and events), and *Grafana* (providing a visual dashboard for service operation). Fig. 7 shows a screenshot of the *Grafana* dashboard we created.
- **Metrics** component manages the calculation of an specific element that is significant for the CA, based on the event data harvested from the cluster.
- **Billing** component includes the parsing and analysis of the CA and the gathering of metrics in order to generate a set of rules and facts that can be computed by the evaluator component to calculate a bill.
- **Evaluator** component encapsulates the rule engine to compute the actual cost and compensation for a given set of rules and facts.
- Finally, the **Render** component provides a user interface (as shown in Fig 8) for the management of princing and SLA terms of the CA. Note that in the screenshot presented, a billing has been generated so that objectives and condition rewards are marked in green if met, red otherwise. All of them can be changed in order to simulate variations in the billing process.

The prototype follows the principles of microservices architectures where each component have a well-defined responsibility and can be evolved and deployed in an independent way; in this context, each component implements a RESTful interface modeled based on the OpenAPI Specification that is enriched with interactive documentation and testing portals generated using the *Swagger-UI* Node.js module.

The microservices paradigm deployed in a Kubernetes platform represents a scalable model that allows the seamless replication of each microservice in order to prevent bottlenecks. For example, in a scenario with a high number of metrics or complex rules the appropriate component could be scaled with multiple instances.

From a metric gathering perspective, as the monitor is based on the *InfluxDB* persistency platform, the current prototype could be easily extended with the usage of *Telegraf* Plugins[4] that provide the mechanism to incorporate metrics from different public cloud providers (such as *AWS*, *Azure* or *Google*) and distributed tracing system implemented with *Zipkin* that can be used to monitor microservice architectures

[4]https://www.influxdata.com/products/integrations/

FIGURE 7. Screenshot of the Grafana dashboard component.

or Function as a Service (Faas) platforms as *Apache OpenWhisk*.

B. INDUSTRIAL VALIDATION RESULTS

Our prototype implementation was developed and deployed within the course of an industrial research project (POET-ISA), whose setting resembled the same motivating scenario described in Sec. II. The service provider that were running the educational platforms service infrastructure tested our proposal directly on the production environment, thanks to the capabilities of the Kubernetes platform to hot deploy the components presented before.

As a result, this deployment provided support to the shift from the pre-existing billing model to a fine-grained, dynamic billing. On the one hand, the pre-existing model was defined as: $MonthlyBill = BaseCost * PlatformCount * ActiveDays$ where *ActiveDays* is independent of the usage of each platform and it just represents the availability of the platform. On the other hand, the evolved billing were tailored with compensations based on the real-time usage of each platform; concretely, the discount model defined is based on three usage metrics in the infrastructure (Memory consumption, CPU load, and disk space) which could potentially result in much competitive billings with up to 75% of total reductions over the pre-existing cost. Figures 7 and 8 show actual data from the industrial validation and the CA applied, correspondingly.[5]

C. PERFORMANCE ANALYSIS

Billing management is typically a crucial task for organizations and, depending on the size and business model, the execution of this process could be challenging. Consequently, in order to analyse the operational limits of our approach, in this section we detail the results of a set of performance experiments to assess the system under different conditions in order to characterise an estimation of its performance in a wide range of scenarios in terms of the following dimensions:

- *D1: Number of metrics used in the CA*. Each metric may correspond with monitored data harvested in the platform, a single artefact measure (direct metric), or an aggregation calculated in terms of other measures (indirect metric).
- *D2: Number of pricing or SLA terms of the CA*. Each term correspond with a guarantee that should be analysed along with a general penalty or reward definition that is derived from the over- or under-usage of the platform, correspondingly.
- *D3: Rules complexity*. The complexity of expressions used in the definition of the billing rules can vary from a simple, fixed threshold (e.g. $TotalAverageCPU < 90$) to multiple combinations of logical operators that result in complex conditions (e.g. $(Node1CPU < TotalAverageCPU) \vee (Node1MEM < TotalAverageMEM)$).
- *D4: Simultaneous billing requests*. In realistic settings multiple billings for different customers should be calculated in parallel. In order to calculate the throughput of the system, it is important to characterise the parallelism limit[6] where the system have a billing generation capacity that provides a stable performance level. We can overcome these operational limits by scaling up the number of microservices on the affected components

[5]The complete CA used in this validation scenario and the InfluxDB dataset generated during the project be found at https://isa-group.github.io/2020-10-billing-lifecycle/

[6]This limit is highly dependent from the underlying infrastructure and deployment mechanism.

(typically the billing and evaluator components) and/or deploying the system in a more advanced instance in an IaaS provider. The study of how the system could perform in different types of IaaS instances is out of the scope of this work.
- *D5: Data points volume per metric*. Different scenarios would require different types of metrics with variable resolutions that represent a particular volume; e.g. a given metric could have higher resolution depending if the monitoring data is harvested every second or every minute.

In the following, we first introduce the infrastructure used in the performance analysis tests, then we discuss the experimental process outlining the different kinds of experiments developed in order to assess the identified dimensions, and finally we present the results of the analysis highlighting the key performance aspects found.

1) TESTBED

In order to define a controlled set of experiments, the developed prototype has been isolated from the production kubernetes platform and deployed separately in a single dedicated machine using a set of docker containers, as opposed to the industrial validation described in Sec. IV-B. Nevertheless, we exported the real dataset of metrics harvested in that scenario, so that we were able to use it for several of the experiments as we detail later in Sec. IV-C2. From an infrastructure perspective, the experimental testbed consists of a single dedicated machine with an Intel Core i7-7700HQ 2.80GHz running Windows 10 with a total 16 GB of RAM. In this host, the prototype were deployed over a Docker desktop platform with a configuration of 2 CPUs and 2 GB of RAM, while the experiment runner (built on a Node.js stack) were configured with a limit of 8 GB of RAM.

Specifically, the experiments ecosystem[7] of the testbed is based on a core runner that generates a CA and metrics data based on a set of parameters, and then requests a billing generation from the prototype, measuring its performance. In this context, the different experiments are defined by means of scripts that parameterize the core runner with a specific configuration. In addition, the experiment runner includes a metric generator which has been designed to complement the real dataset to analyse the capabilities of the platform concerning *D1* dimension.

Specifically, the configuration input for the core runner can be grouped in three sets as follows:

- *Customer agreement generation parameters*. These parameters include the number of terms for the CA to be generated (metrics, pricing, guarantees, and compensations) and the rule set to generate the guarantee objective expression. Currently, three different complexity generation rules are available (simple, medium, and complex) ranging from simple conditions that contains a single comparison operator with literals, to complex aggregated expressions combining logical, algebraic, and comparison operators.
- *Metrics generation parameters*. These parameters guide the generation of the data points that will be inserted in the metrics database that serves as a source of information for the billing generation. The interval of the metrics data points generated corresponds with the same interval of the real dataset monitored in the industrial validation (i.e. 13 months) in order to have a comparable context, though the resolution (data points per second) is parameterized to allow different data point densities.
- *Invocation parameters*. These parameters include the iteration number and parallelism used to invoke the prototype allowing a repetition of calculations in a sequential and parallel model.

[7]The complete list of scripts and datasets generated for the different experiments are available at https://github.com/isa-group/billing-performance

2) EXPERIMENTAL PROCESS

In order to assess the performance of our approach amongst the different identified dimensions, we designed an experimental study. Specifically, for each dimension, we performed a corresponding experiment using the testbed and adjusting the core runner parameters accordingly with different configurations that explore those dimensions. Thus, each experiment run (using a specific configuration) is repeated a number of times (100 in our case) in order to calculate the average and standard deviation, and to assure a statistical significance of the results. Specifically, we developed 5 successive experiments with 43 different configurations (for a total duration of over 20 hours considering experimental setup and runtime):

- *Exp1*. In order to explore *D1* dimension, we used 10 different configurations to generate CAs with a number of metrics ranging from 1 to 1024 following a growth factor of 2.
- *Exp2*. With respect to *D2*, we used 10 different configurations to generate CAs with a number of guarantee terms ranging from 1 to 1024 following a growth factor of 2.
- *Exp3*. In order to consider *D3* impact, we used 3 different configurations to generate CAs with different complexity degrees for guarantee objectives.
- *Exp4*. For exploring *D4*, we used 10 different configurations to assess the behaviour with different scenarios of parallel billing generations, ranging from 1 to 10 concurrent users.
- *Exp5*. In order to explore *D5*, we set up 10 different configurations of data point densities per metric, starting with 6 data points and following a growth factor of 3.

3) EXPERIMENTAL RESULTS

The summary of the results of each experiments are depicted in Figure 9. As a global conclusion, we can identify a linear correlation between the number of CA terms and the performance obtained while there is no correlation with the data points density for each metric or the complexity of terms.

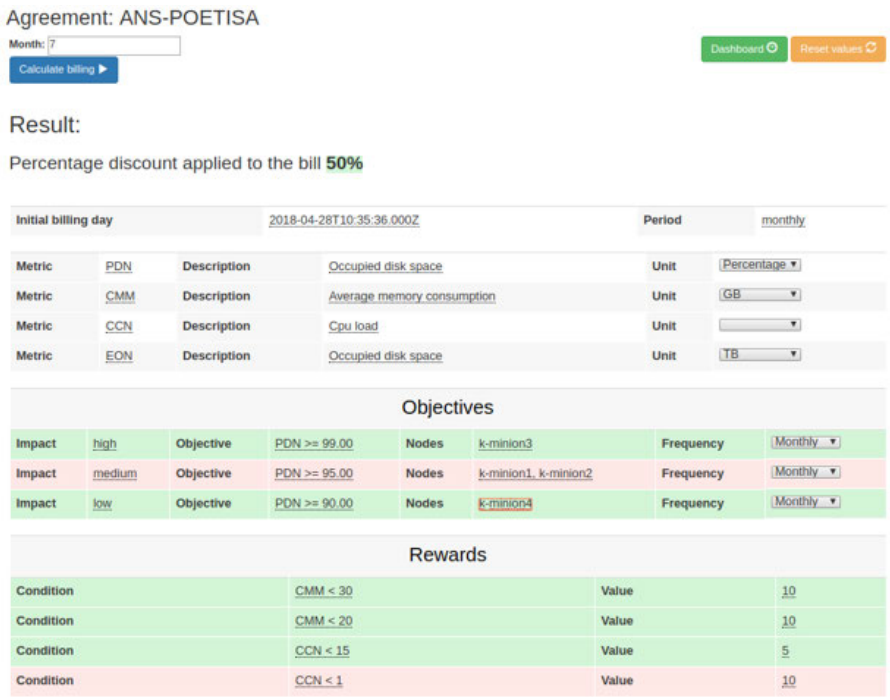


FIGURE 8. Screenshot of the Render component.

Specifically, as shown in Fig. 9, the obtained performance presented a 2nd-order polynomial and linear proportion with the size of the CA depending on the number of metrics (*Exp1*) and the number of guarantee terms (*Exp2*), correspondingly. Thus, our solution can manage a sufficiently high number of metrics and terms without incurring into performance issues for billing generation.

Concerning the expression complexity (*Exp3*), Fig. 9 shows it does not impact the overall performance, since average values are close to 1 second, within approximately 100 ms of deviation in the different runs of our experiment (according to the error bars shown). This phenomenon can be due to the fact that expression evaluation is performed by the native runtime of the rule engine that provides a highly efficient model.

Concerning the parallel invocations of billing calculations (*Exp4*), the system scales in a linear proportion without overlapping effects so we could forecast the expected behaviour for a given estimated rate of requests per second using a direct interpolation.

In a production environment, *Exp4* should be repeated to characterise the baseline performance of the production infrastructure. Since the whole components are based on Node.js stack, the core computation model is based on a non-blocking, event driven I/O. Consequently, while gaining a high level of throughput, each microservice operates on a single-thread that prevents the parallelism of computations. This computation model is effective in our scenario since the different calculations are fragmented in different microservices and most of them doesn't imply CPU intensive tasks.

However, it is important to highlight that the characterisation of the operational limits of a particular deployment is crucial: in our performance experimental study (based on the testbed configuration described in section IV-C1) we have observed a linear response behaviour while the number of CA terms/metrics are under a certain threshold (around 512 metrics / terms). In turn, once past this threshold, computations on the different microservices overlap and the performance quickly degrades. In order to achieve a higher performance in more demanding scenarios, we can use, for example, a more performant infrastructure (in terms of CPU) or Docker configuration (in our experiments we used a cap limit of 2GB of RAM and 2 cores). Moreover, in the specific case that the single thread architecture of Node.js represents an important penalty for the concurrent usage of a given stressed microservice, the container model allows the scalability of several instances of the stressed microservice, incorporating a load balancing element that would represent a cluster that accept parallel requests overcoming the single thread limitation.

Concerning Exp5, Fig. 9 shows the data point density of the dataset generated does not have an impact in the performance of the system, with an average response time below 1 second and a standard deviation under 100 ms, according to our experiments. Consequently, since we rely on the query mechanisms of the underlying *InfluxDB* time series database, our system can provide a proper throughput of billing generations in scenarios with vast numbers of data points.

D. ANALYSIS OF INDUSTRIAL TOOLING

In addition to the aforementioned prototype development and industrial validation, we also analysed the support to the billing life cycle of commercial tools, providing a comparison framework. The analysed tools were obtained using

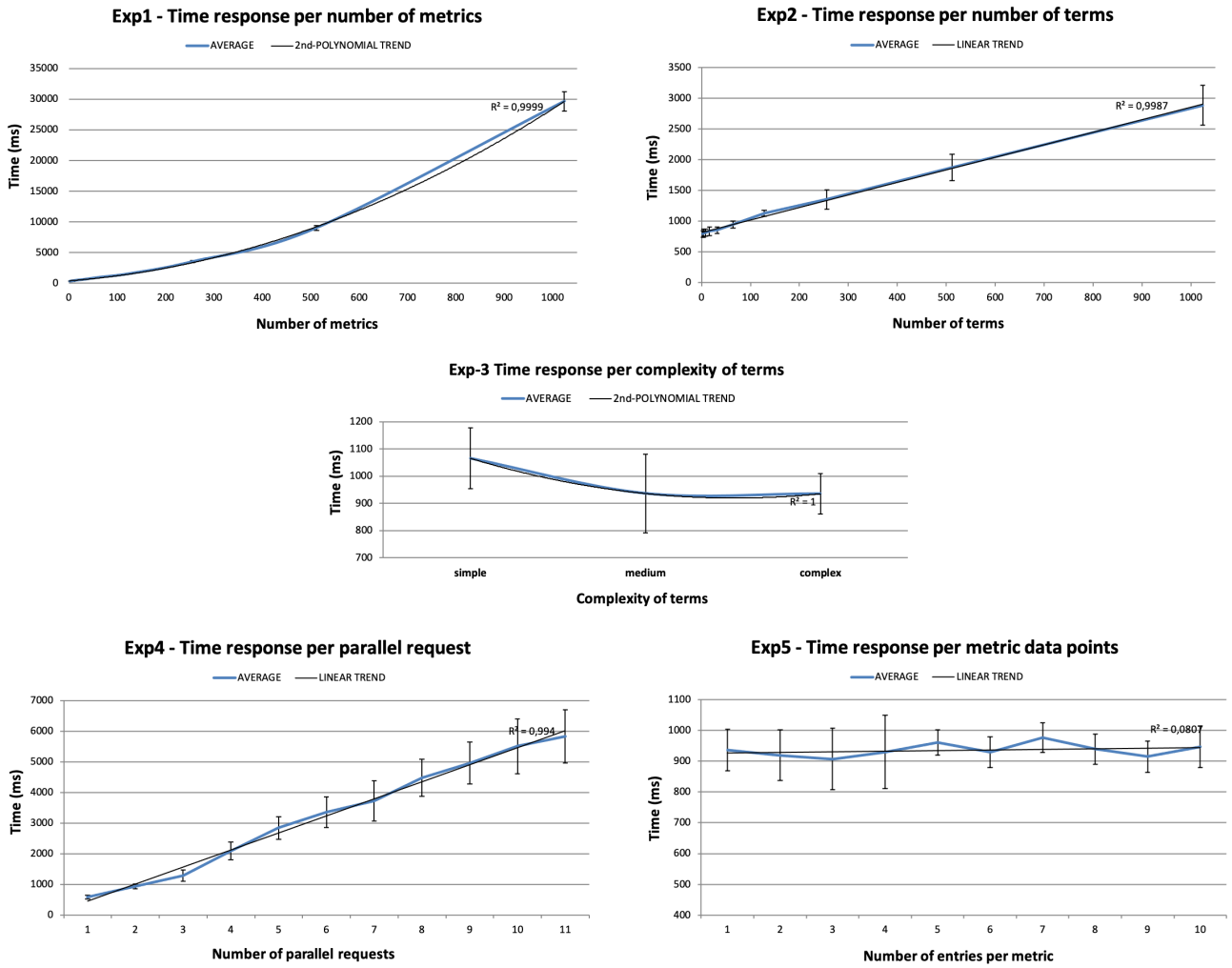


FIGURE 9. Summary of the five experiments results.

Capterra[8] and Software Advice[9] web sites, which are devoted to software searching. The keywords used for the queries were "*billing and invoicing*". Additional filters restricted the selection to tools used by more than 1000 users. Finally, we included in our study 53 tools, which support cloud-based billing and offer a pricing model. Table 1 showcases our study results in alphabetical order,[10] while Fig. 10 sums up the comparison, where we highlight the percentage of tools which support each analysed feature. Noteworthy, our prototype tool supports most of these features, except for real-time forecasting. In the following, we discuss our findings, comparing the capabilities of industrial tools and our proposal when addressing the challenges identified in Sec. II-B

1) SPECIFYING THE PRICING MODEL

First, we analyse how industrial tools address the first challenge we identified in Sec. II-B, focusing on the facilities they provide to specify pricing models. Providers are mostly interested in monetising their cloud infrastructure to obtain revenues. This includes one or more of the following operations and features:

- Selecting a proper *pricing model* including subscription plans and *discount policies*. This is usually carried out by means of forms and templates.
- *Quoting products* according to the pricing model. As an example, in cloud context, the *rating* is a kind of quoting, consisting of setting the base prices per unit for services and resources.
- Configure a *catalog of products*, both services and resources.

Catalogs configuration, pricing and discounts, and quoting together are known as *CPQ management* [10], a key aspect in the context of revenue management. There are 15 tools that offer CPQ-like operations, namely `Biltrix24`, `ChargeOver`, `HarmonyPSA`, `HostBill`, `jBilling`, `Jerasoft`, `Neon`, `OneBill`, `Oracle's Financial Cloud`, `PandaDoc`, `Salesboom`, `Salesforce`, `SpryBill`, `Unicorn Billing`, and `Zuora`.

[8] https://www.capterra.es/
[9] https://www.softwareadvice.com/
[10] The unabridged version of our initial study can be found at https://isa-group.github.io/2020-10-billing-lifecycle/

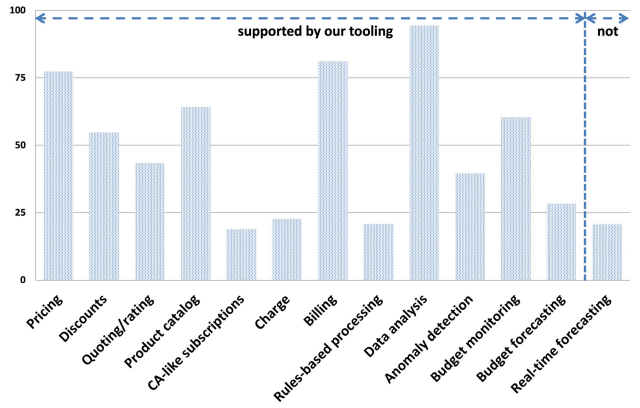


FIGURE 10. Summary of industrial tooling support.

In comparison to our solution, it also offers CPQ-like operations, since one can define different CAs offering customised pricing and billing models. Furthermore, CAs are fundamental in our approach, leveraging them as first-class citizen for governing the cloud infrastructures.

2) SUPPORTING THE BILLING LIFE CYCLE

With regards to the billing specification, customisation and support of the subprocesses involved, we focus our analysis in four characteristics related to the second challenge identified in Sec. II-B. First, in order to control the infrastructure usage, most providers have a subscription management for consumers. These subscriptions usually consist of allowing subscribed consumers to use the infrastructure, but no more. Only 10 of the analysed providers offer *CA-like subscriptions* which regulate these relationships, namely `Billwerk`, `BluBilling`, `GoodSign`, `HarmonyPSA`, `Oracle's Financial Cloud`, `PandaDoc`, `Salesboom`, `Salesforce`, `Soft-ex Platform`, and `SpryBill`. In turn, our proposal treats CAs as main artefacts, including the subscription data together with specific pricing and billing options, as we previously discussed.

Once the infrastructure is being used, providers are also interested in the billing generation of invoices for consumers to pay. This process typically consists of two subprocesses:

- *Charging*, where the usage records are evaluated to obtain the corresponding charges, applying the applicable rating.
- *Billing*, where the charge records are collated and the invoice is generated.

From the analysed tools, 12 of them offer charging and billing features, such as `Amdocs Optima`, `BluBilling`, `Chargify`, `Fusebill`, `GoodSign`, `HostBill`, `jBilling`, `OneBill`, `Oracle's Financial Cloud`, `SpryBill`, `TimelyBill`, and `Unicorn Billing`. Our proposal also supports both charging and billing, by means of the CAs established between consumers and providers.

Most analysed tools provide forms and/or templates to specify and customise the billing life cycle. However, this is somehow limited, since their underlying models remain hidden, or they are not editable. Rules allow to improve customisation to a greater extent. As providers may have different business and pricing models, this variability may be tackled with rules configuring different discounts, penalties, and other business conditions. Although some tools offer pre-defined rules with *ad hoc* processing, most advanced tools have a rule-specific language, including complex conditions involving flexible metrics and scoping, and allowing complex actions to be executed in response to conditions. These rules are processed by rule engines integrated in cloud infrastructures. Concretely, there are 11 tools that offer advanced rules, namely `BluBilling`, `GoodSign`, `Invoiced`, `Neon`, `OneBill`, `Oracle's Financial Cloud`, `Salesforce`, `SpryBill`, `Subscription DNA`, `Unicorn Billing`, and `Zuora`. In turn, our proposal allows providers to include advanced rules in pricing and billing terms of CAs, enabling customer personalization by means of instantiating the proper rules when creating the agreement.

There are other features and subprocesses in the billing life cycle, such as showback, chargeback, payment options, integration with payment gateways, and accountability, which are out of the scope of our comparison framework.

3) DASHBOARDS

Finally, we analyse how industrial tools tackle the third challenge we identified in Sec. II-B, with regards to how they facilitates the automation of the different activities discussed in Sec. III-A.

Consumers are interested in using services and resources from providers, as cheaply and efficiently as possible, while checking that the billing generation is correct. Usually, tools aimed at consumers consist of a dashboard which collects data from multiple sources, requiring transparency from their providers, the integration of third-party services, and the use of time-series databases. Once data is collected, these dashboards offer a set of *data analytic operations* for supporting customers to understand the billing validation or billing estimation, such as usage trends, *anomaly detection*, *budget monitoring*, and *budget forecasting*. In addition to data analytics, these tools may also include different comparisons among providers to support consumers in decision making.

In turn, in order to help providers in managing the billing life cycle, tools also include additional data analytics specific to them, especially regarding to billing, budget monitoring and forecasting.

As shown in Table 1, there are 12 tools providing data analytics, anomaly detection, together with budget monitoring or forecasting, i.e. `Billwerk`, `Chargebee`, `Cloudability`, `Flexera's FlexNet Optima`, `GoodSign`, `HarmonyPSA`, `Neon`, `OneBill`, `Opencell Billing`, `Soft-ex Platform`, `SpryBill`, and `Unicorn Billing`. However, only 11 tools offer real-time forecasting, such as `Cloudkick`, `Flexera's FlexNet Operations`, `Hyperic`, `Jerasoft`,

TABLE 1. Industrial tools supporting the billing life cycle.

	Pricing model specifications				Billing life cycle specifications				Dashboard features				
Tool name	Pricing	Discounts	Quoting / rating	Product catalog	CA-like subscriptions	Charge	Billing	Rules-based processing	Data analytics	Anomaly detection	Budget monitoring	Budget forecasting	Real-time forecasting
2Checkout	✓		✓				✓		✓		✓	✓	
Amdocs Optima	✓		✓	✓		✓	✓		✓				
Aspect Via (R) Dashboard									✓				
Billingbooth	✓			✓			✓		✓		✓		
Billwerk	✓	✓		✓	✓		✓		✓	✓	✓	✓	
Bitrix24	✓	✓	✓	✓			✓		✓		✓		
BluBilling	✓	✓	✓		✓	✓	✓	✓	✓		✓		
Chargebee	✓	✓		✓			✓		✓	✓	✓	✓	
ChargeDesk	✓	✓		✓			✓				✓	✓	
ChargeOver	✓	✓	✓	✓			✓		✓		✓	✓	
Chargify	✓	✓		✓		✓	✓		✓		✓		
Cloudability									✓	✓	✓	✓	
Cloudkick									✓	✓			✓
Embotics' vCommander	✓						✓		✓				
Flexera's FlexNet Operations	✓			✓			✓		✓	✓			✓
Flexera's Optima									✓	✓		✓	
Fusebill	✓	✓		✓		✓	✓		✓		✓	✓	
GoodSign	✓			✓	✓	✓	✓	✓	✓	✓	✓	✓	
HarmonyPSA	✓	✓	✓	✓	✓		✓		✓	✓	✓	✓	
HostBill	✓	✓	✓	✓		✓	✓						
Hyperic									✓	✓			✓
Invoice Ninja	✓		✓	✓			✓				✓		
Invoiced	✓	✓	✓				✓	✓	✓				
jBilling	✓	✓	✓	✓		✓	✓		✓		✓		
Jerasoft	✓	✓	✓	✓			✓		✓		✓		✓
Monitis									✓	✓			✓
Neon	✓	✓	✓	✓			✓	✓	✓	✓	✓	✓	
Nimsoft									✓	✓			✓
OneBill	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓		
Opencell Billing	✓	✓		✓			✓		✓	✓	✓		
Oracle's Financial Cloud	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓		✓
Orbitera's Ecosystem	✓						✓		✓				
Opencloudware's Platform							✓		✓				
Pabbly	✓	✓		✓			✓		✓		✓		
PandaDoc	✓	✓	✓	✓	✓		✓		✓				
PayPal Invoicing	✓			✓			✓		✓		✓	✓	
Recurly	✓	✓					✓		✓		✓		
Sage	✓	✓	✓				✓		✓		✓		
Salesboom	✓	✓	✓	✓	✓		✓		✓		✓	✓	
Salesforce	✓	✓	✓	✓	✓		✓	✓	✓	✓			✓
SDK.finance	✓	✓		✓	✓		✓		✓		✓		
Soft-ex Platform				✓			✓		✓	✓	✓		
SpryBill	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	
Stripe Billing	✓	✓		✓			✓		✓		✓		
Subscription DNA	✓	✓					✓	✓	✓				
Tableaux's IT Analytics									✓	✓			
Telco Billing Solution	✓		✓	✓			✓		✓		✓		
TimelyBill	✓		✓	✓		✓	✓		✓	✓			✓
Unicorn Billing	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓		
Waldur's Dashboard							~		✓				✓
Z-Billing	✓			✓			✓		✓				
Zabbix									✓	✓			✓
Zuora	✓	✓	✓	✓			✓	✓	✓		✓	✓	
Our proposal	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	

Monitis, Nimsoft, Oracle's Financial Cloud, Salesforce, TimelyBill, Waldur's Dashboard, and Zabbix.

Our developed tooling provides both a dashboard, which renders the CA using actual metrics and pricing values, highlighting anomalies whenever they may appear, as well as a customisable representation of the CA containing the billing rules defined, as showcased in Figures 7 and 8, correspondingly. In our proposal, we focus on current data metrics from actual running services, but we do not support billing forecasting.

V. RELATED WORK

We organise our study of the related work in two perspectives, one analysing works on pricing models, and the other analysing works related with the billing process.

A. ON PRICING MODELS

Several comparative analysis that can be found in the literature [11]–[13] show the evolution of pricing models adopted by cloud providers. The main conclusion they observe is that these models vary from static subscriptions to much more dynamic and complex schemes applying variants of pay-on-demand, subscription, and auction-based pricing. Some authors propose the addition of trust models to verify billable events in the cloud [14], [15]. On the one hand, Park *et al.* in [14] propose THEMIS, a secure and non-obstructive billing system using a cloud notary authority to generate mutually verifiable binding information that can be used to resolve efficiently potential disputes between an user and a cloud service provider. On the other hand, Chen *et al.* in [15] present BitBill, a scalable, robust, and mutually verifiable billing system to ensure a natural and intuitive peer-to-peer verification of billable events in the cloud, leveraging a Bitcoin-like mechanism. Although THEMIS includes a component to monitor SLAs, neither THEMIS nor BitBill provide, as far as we know, support for specifying discounts and other elements included in our proposed pricing model.

In a recent work [16], Zhang *et al.* provides an OWL-based ontology defining pricing and performance features of IaaS providers. However, although they mention a way to describe price exceptions in natural language (e.g. special rate when network traffic egress between zones in the same region) they do not provide a way to specify compensations or discounts as defined in our proposal. Other recent works are focusing on defining pricing models considering resources allocation optimisation mechanisms, as in [1], where Alzhouri *et al.* investigate a dynamic pricing for re-using stagnant virtual machines. Specifically, their approach manages multiple classes of virtual machines in order to achieve the maximum expected revenue within a finite discrete time horizon.

Similarly, in [2] Lai *et al.* propose to tackle the fee-setting problem of cloud providers by means of a chance-constrained bi-level optimisation model, where the fee-setter acts as a leader, and all the clients act as followers who make decisions based on the price they receive. A similar problem is tackled by Roy *et al.* [17] by proposing PRIME, a pricing scheme supporting an optimised and unbiased distribution of profit among different actors in mobile sensor-cloud architectures. Yao *et al.* [3] provide an optimal overbooking policy to maximize providers' profits and enhance cloud users' experiences by decreasing the number of SLA violations. They achieve these goals by establishing a cooperative game model of providers organised in a cloud federation with sharing resources, and making a reasonable profit distribution. Such a technique has proved to be applicable to optimise pricing and user experience in data-centric computing frameworks by Huang *et al.* [18]. In turn, Zhang *et al.* [19] design an online auction for dynamic resource scaling and pricing, where cloud users repeatedly bid for resources into the future with increased amounts, according to their scale-up/out preferences.

Other authors, such as Tortonesi and Foschini [20], explore an approach using genetic algorithm to dynamically reconfigure IT service components placement, in order to respond to pricing changes, and to control and guarantee the SLAs defined by service providers. Finally, in [21], Vinu Prasad *et al.* explore the problem of cloud users procuring arbitrary bundles of resources from cloud providers. They propose a combinatorial auction mechanism in which users submit their requirements, and in turn vendors submit bids containing their offered bundles of resources, price, and quality of service.

However, only some of the aforementioned authors consider the underlying SLA model in their proposals [3], [20]. As far as we know, these works do not allow describing billing discounts as opposed to our proposal, even though they are commonly specified in customer agreements by cloud providers.

B. ON BILLING PROCESS

Focusing on proposals related to the billing process and life cycle, several works can be found [7], [22]–[27]. Among them, one of most relevant is CYCLOPS [22]. This framework has been developed in the context of T-NOVA project, funded by the European Commission (EC). It proposes a rule-based process to develop a dynamic rating, charging, and billing for cloud service providers. CYCLOPS includes an SLA module to get the violation conditions and penalties to be applied during the billing. The main difference with our proposal is the central role of CAs that in our case includes both, SLA terms and all kind of rules for billing governance. Thus, the billing process can be customised by means of instantiating specific rules for each customer agreement to be created. In contrast, rules in CYCLOPS are inserted into the framework by their administrators, so that rules govern the billing process, but they are not specific for each customer and cannot be dynamically adapted.

There are other proposals related to the billing process, though, as far as we know, they do not support the billing customisation facilities that our proposal provides. Thus, in [23], [24], Elmroth *et al.* introduce a billing

and accounting architecture for federated clouds, taking into account the time-varying resources, migrating services between datacenters, and different pricing schemes. In [28] Lindner *et al.* propose a cloud supply chain relying on an underlying monitoring framework combined with a billing and accounting framework, both based on a common information model. These works have been developed in the context of RESERVOIR project, funded by the EC, too. In turn, Harsh *et al.* [25] introduce a uniform API, billing, and monitoring features in SLA-governed, federated clouds, so that resources from different providers can be integrated seamlessly as a single, homogeneous cloud. It has been developed in the context of CONTRAIL EC funded project.

Apart from billing generation, which is the main concern of our work, there are also proposals focused on validation and estimation. For instance, in [7] Mihoob *et al.* introduce improvements on accounting models for providers to make easier both billing validation and estimation, so avoiding discrepancies between providers and customers. It applies on Amazon Simple Storage Service (S3) and Elastic Compute Cloud (EC2). Zarnekow [26], introduce a pricing and accounting model for customers to know the impact of adopting a cloud infrastructure, and gives a comparison among the most usual providers. Finally, in [27], Cho *et al.* propose an IaaS cost estimation model that enables real-time cost monitoring. By knowing in advance such estimated costs, consumers may avoid the higher cost of performing a continuous monitoring.

VI. CONCLUSION

Service billing customisation is a challenging task for cloud providers that need to adapt their pricing and discount rules dynamically, according to business goals, customer agreements, and their corresponding SLAs. Nevertheless, service consumers can also benefit from customised billing policies, adapted to their actual needs. Our proposal focus on the definition of said policies, which results in an integrated monitoring and billing process. From a set of customisable metrics and billing rules, our solution generates the corresponding bills, while also considering CAs monitoring and enforcement.

As shown in our validation scenario, by means of a customised service billing we can adjust generated bills to achieve both consumers and providers goals. Thus, our proposed solution allows them to define highly-expressive customer agreements including SLAs and pricing terms. These pricing terms support specifying discounts based on a set of service metrics. We have validated the proposal by developing an automated tool used in an industrial scenario providing a monitoring dashboard to analyse the suitability of the policies and obtain evidences to sustain the generated billing information. Furthermore, we have evaluated the tool performance by means of 5 experiments that evaluates its response time under different conditions of load and CAs expressiveness.

Finally, we have analysed 53 industrial tooling that support pricing and billing processes. As a result, we conclude that none of the analysed tool support all the features of our proposed solution, which addresses the three key challenges we identified, namely the specification of pricing and billing related information within the CA, the use of highly expressive models to describe billing rules, and the automation of the billing generation process based on said models.

As future work, we will research on how our proposed model can be applied to billing validation and estimation, which are related processes mainly relevant to service consumers. In addition, since terms as Customer Agreements, Agreements, SLAs, are used by providers but with some slight semantic differences, a further interesting research may include a unified model for all agreement-related terms. Moreover, an interesting potential evolution of the current work would be to incorporate prediction techniques based on Machine Learning to provide useful dashboards for both providers and consumers in order to forecast their future bills and understand the pricing tendencies.

ACKNOWLEDGMENT

The authors would like to thank Ibone González and Pablo García for their support on the prototype implementation and evaluation.

REFERENCES

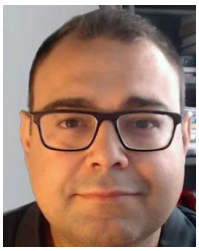
[1] F. Alzhouri, A. Agarwal, and Y. Liu, "Maximizing cloud revenue using dynamic pricing of multiple class virtual machines," *IEEE Trans. Cloud Comput.*, early access, Oct. 25, 2018, doi: 10.1109/TCC.2018.2878023.
[2] C. Lai and L. Xu, "Bilevel fee-setting optimization for cloud monitoring service under uncertainty," *IEEE Access*, vol. 6, pp. 9473–9483, 2018.
[3] M. Yao, D. Chen, and J. Shang, "Optimal overbooking policy for cloud service providers: Profit and service quality," *IEEE Access*, vol. 7, pp. 96132–96147, 2019.
[4] K. P. Joshi and C. Pearce, "Automating cloud service level agreements using semantic technologies," in *Proc. IEEE Int. Conf. Cloud Eng.*, Piscataway, NJ, USA: Institute of Electrical and Electronics Engineers, Mar. 2015, pp. 416–421.
[5] J. M. García, P. Fernández, C. Pedrinaci, M. Resinas, J. Cardoso, and A. Ruiz-Cortés, "Modeling service level agreements with linked USDL agreement," *IEEE Trans. Services Comput.*, vol. 10, no. 1, pp. 52–65, Jan. 2017.
[6] C. Müller, A. M. G. Fernandez, P. Fernandez, O. Martín-Díaz, M. Resinas, and A. Ruiz-Cortes, "Automated validation of compensable SLAs," *IEEE Trans. Services Comput.*, early access, Dec. 7, 2018, doi: 10.1109/TSC.2018.2885766.
[7] A. Mihoob, C. Molina-Jimenez, and S. Shrivastava, "Consumer side resource accounting in the cloud," in *IFIP Advances in Information and Communication Technology*, vol. 353. Berlin, Germany: Springer, 2011, pp. 58–72.
[8] J. M. García, O. Martín-Díaz, P. Fernandez, A. Ruiz-Cortés, and M. Toro, "Automated analysis of cloud offerings for optimal service provisioning," in *Service-Oriented Computing* (Lecture Notes in Computer Science), vol. 10601. Berlin, Germany: Springer, 2017, pp. 331–339.
[9] F. Hayes-Roth, "Rule-based systems," *Commun. ACM*, vol. 28, no. 9, pp. 921–932, 1985.
[10] K. Sorri, M. Kumpulainen, M. Seppänen, M. Dunne, and K. Huittinen, "Prospects of CPQ: Evolving toward industry platforms," Tech. Rep.
[11] M. Al-Roomi, S. Al-Ebrahim, S. Buqrais, and I. Ahmad, "Cloud computing pricing models: A survey," *Int. J. Grid Distrib. Comput.*, vol. 6, no. 5, pp. 93–106, Oct. 2013.
[12] S. Kansal, G. Singh, H. Kumar, and S. Kaushal, "Pricing models in cloud computing," in *Proc. Int. Conf. Inf. Commun. Technol. Competitive Strategies (ICTCS)*. New York, NY, USA, Oct. 2014, pp. 1–5.

[13] A. Mazrekaj, I. Shabani, and B. Sejdiu, "Pricing schemes in cloud computing : An overview," *Int. J. Adv. Comput. Sci. Appl.*, vol. 7, no. 2, pp. 80–86, 2016.
[14] K.-W. Park, J. Han, J. Chung, and K. H. Park, "THEMIS: A mutually verifiable billing system for the cloud computing environment," *IEEE Trans. Services Comput.*, vol. 6, no. 3, pp. 300–313, Jul. 2013.
[15] L. Chen and K. Chen, "BitBill: Scalable, robust, verifiable peer-to-peer billing for cloud computing," in *Proc. 6th USENIX Conf. Hot Topics Cloud Comput. (HotCloud)*, 2014, p. 20.
[16] Q. Zhang, A. Haller, and Q. Wang, "CoCoOn: Cloud computing ontology for IaaS price and performance comparison," in *The Semantic Web—ISWC 2019*. Cham, Switzerland: Springer, 2019, pp. 325–341.
[17] A. Roy, S. Misra, and S. Nag, "PRIME: An optimal pricing scheme for mobile sensors-as-a-service," *IEEE Trans. Mobile Comput.*, early access, Sep. 14, 2020, doi: 10.1109/TMC.2020.3023885.
[18] M. Huang, W. Liu, T. Wang, Q. Deng, A. Liu, M. Xie, M. Ma, and G. Zhang, "A game-based economic model for price decision making in cyber-physical-social systems," *IEEE Access*, vol. 7, pp. 111559–111576, 2019.
[19] X. Zhang, Z. Huang, C. Wu, Z. Li, and F. C. M. Lau, "Dynamic VM scaling: Provisioning and pricing through an online auction," *IEEE Trans. Cloud Comput.*, vol. 9, no. 1, pp. 131–144, Jan. 2021.
[20] M. Tortonesi and L. Foschini, "Business-driven service placement for highly dynamic and distributed cloud systems," *IEEE Trans. Cloud Comput.*, vol. 6, no. 4, pp. 977–990, Oct. 2018.
[21] G. V. Prasad, A. S. Prasad, and S. Rao, "A combinatorial auction mechanism for multiple resource procurement in cloud computing," *IEEE Trans. Cloud Comput.*, vol. 6, no. 4, pp. 904–914, Oct. 2018.
[22] S. Patanjali, B. Truninger, P. Harsh, and T. M. Bohnert, "CYCLOPS: A micro service based approach for dynamic rating, charging & billing for cloud," in *Proc. 13th Int. Conf. Telecommun. (ConTEL)*, Jul. 2015, pp. 1–8.
[23] E. Elmroth, F. G. Márquez, D. Henriksson, and D. P. Ferrera, "Accounting and billing for federated cloud infrastructures," in *Proc. 8th Int. Conf. Grid Cooperat. Comput.*, Aug. 2009, pp. 268–275.
[24] B. Rochwerger, D. Breitgand, E. Levy, A. Galis, K. Nagin, I. M. Llorente, R. Montero, Y. Wolfsthal, E. Elmroth, J. Caceres, M. Ben-Yehuda, W. Emmerich, and F. Galan, "The reservoir model and architecture for open federated cloud computing," *IBM J. Res. Develop.*, vol. 53, no. 4, pp. 1–4, 2009.
[25] P. Harsh, Y. Jegou, R. G. Cascella, and C. Morin, "Contrail virtual execution platform challenges in being part of a cloud federation," in *Towards a Service-Based Internet-ServiceWave*. Berlin, Germany: Springer, 2011, pp. 50–61.
[26] R. Zarnekow, "The impact of cloud computing adoption on IT service accounting approaches—A customer perspective on IaaS pricing models," Ph.D. dissertation, Dept. Inf. Commun. Manage., Teschnische Universität Berlin, Germany, 2012.
[27] K. Cho and H. Bahn, "Real-time IaaS cost model and estimation results for cloud PC laboratory service," in *Proc. Int. Commun. Eng. Cloud Comput. Conf.*, Oct. 2019, pp. 14–18.
[28] M. Lindner, F. Galán, C. Chapman, S. Clayman, D. Henriksson, and E. Elmroth, "The cloud supply chain: A framework for information, monitoring, accounting and billing," in *Proc. 2nd Int. ICST Conf. Cloud Comput. (CloudComp)*, Oct. 2010, pp. 1–22.

JOSÉ MARÍA GARCÍA received the M.Sc. degree (Hons.) in computer science and software engineering and the Ph.D. degree in technology and software engineering from the University of Seville, Spain, in 2008 and 2012, respectively.

From 2006 to 2013, he was a Research Assistant with the University of Seville. From 2013 to 2015, he was a Postdoctoral Researcher and an Assistant Professor with the Semantics Technology Institute, University of Innsbruck, Austria. He rejoined the University of Seville, as an Assistant Professor, in 2015. Since 2019, he has been an Associate Professor with the Smart Computer Systems Research and Engineering Laboratory, Applied Software Engineering Group, Research Institute of Computer Engineering, University of Seville. His research interests include blockchain, semantic web technologies, service-oriented architectures, and linked data.

OCTAVIO MARTÍN-DÍAZ received the M.Sc. and Ph.D. degrees in computer science from the University of Seville, Spain, in 1994 and 2007, respectively. He is currently a Lecturer of software engineering with the University of Seville. He is also with the Applied Software Engineering Group and the Research Institute of Computer Engineering. His current research interests include purchasing, pricing, and billing in cloud computing and service-oriented computing, mainly focusing on aspects related to time management in service level agreements.



PABLO FERNANDEZ received the Ph.D. degree in technology and software engineering from the University of Seville, Spain, in 2013. Since 2019, he has been an Associate Professor with the Smart Computer Systems Research and Engineering Laboratory, Applied Software Engineering Group, Research Institute of Computer Engineering, University of Seville. His current research interest includes automated governance of organizations and infrastructures based on service level agreements and commitments.

CARLOS MÜLLER received the Ph.D. degree in technology and software engineering from the University of Seville, Spain, in 2013. He is currently a Lecturer of software engineering with the University of Seville. He is also with the Applied Software Engineering Group and the Research Institute of Computer Engineering. His current research interest includes the automated analysis of SLAs and the application of such analysis at SLA design and monitoring.

ANTONIO RUIZ-CORTÉS (Member, IEEE) received the B.Sc. degree in computer science and the M.Sc. and Ph.D. degrees in informatics engineering from the University of Sevilla, Spain, in 1992, 1996, and 2002, respectively.

From 1990 to 1998, he was in the Computer Industry. From 1996 to 1998, he was a part-time Lecturer with the University of Huelva. He joined the University of Sevilla, as a full-time Lecturer, in 1998. He founded the Applied Software Engineering Group, University of Sevilla, in 2004. Since 2016, he has been a Full Professor of software and service engineering. Since 2019, he heads the SCORE Laboratory, University of Sevilla. His current research interests include service-oriented computing, business process management, and testing and software product lines.

Dr. Ruiz-Cortés is an Elected Member of the Academy of Europe. He was a recipient of the Most Influential Paper of SPLC, in 2017, and the VAMOS Award, in 2020. He received the Extraordinary Award for his B.Sc., M.Sc., and Ph.D. degrees. Since 2018, he has been the President of the Spanish Society on Software Engineering (SISTEDES). He is also an Associate Editor of *Computing* (Springer) and the *International Journal of Cooperative Information Systems*.