IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received January 29, 2021, accepted March 24, 2021, date of publication April 6, 2021, date of current version April 16, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3071417

Empirical Evaluation of a Method for Monitoring Cloud Services Based on Models at Runtime

PRISCILA CEDILLO[1], EMILIO INSFRAN[2], SILVIA ABRAHÃO[2], AND JEAN VANDERDONCKT[3]

[1]Department of Computer Science, Universidad de Cuenca, Cuenca 010203, Ecuador
[2]Department of Computer Science and Computation, Universitat Politècnica de València, 46022 Valencia, Spain
[3]Louvain School of Management, Université Catholique de Louvain, 1348 Louvain-la-Neuve, Belgium

Corresponding author: Silvia Abrahão (sabrahao@dsic.upv.es)

This work was supported by the Spanish Ministry of Science, Innovation, and Universities through the Adapt@Cloud Project under Grant TIN2017-84550-R, and by the Generalitat Valenciana under the UX-Adapt Project, Grant AICO/2020/113.

ABSTRACT Cloud computing is being adopted by commercial and governmental organizations driven by the need to reduce the operational cost of their information technology resources and search for a scalable and flexible way to provide and release their software services. In this computing model, the Quality of Services (QoS) is agreed between service providers and their customers through Service Level Agreements (SLA). There is thus a need for systematic approaches with which to assess the quality of cloud services and their compliance with the SLA. In previous work, we introduced a generic method for Monitoring cloud Services using models at RunTime (MoS@RT), which allows the monitoring requirements or the metric operationalizations of these requirements to be changed at runtime without the modification of the underlying infrastructure. In this paper, we present the design of a monitoring infrastructure that supports the proposed method with its instantiation to a specific platform and reports the results of an experiment carried out to evaluate the perceived efficacy of 58 undergraduate students when using the infrastructure to configure the monitoring of cloud services deployed on the Microsoft Azure platform. The results show that the participants perceived MoS@RT to be easy to use, useful, and they also expressed their intention to use the method in the future. Although further experiments must be carried out to strengthen these results, MoS@RT has proved to be a promising monitoring method for cloud services.

INDEX TERMS Cloud computing, models@runtime, quality of service (QoS), services monitoring, software as a service (SaaS).

I. INTRODUCTION

Cloud Computing is a model that enables ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction [1]. Companies currently consider this model to be an efficient way in which to reduce the operational costs of their information technology resources. Software as a Service (SaaS) uses a cloud computing infrastructure to enable the customer to use the provider's applications running on the Cloud [1]. SaaS applications should satisfy quality characteristics during their provision, which are expressed in Service Level Agreements (SLAs). SLAs are contracts that define the minimal guarantees that a cloud provider should offer to its customers [2]. Penalties are established when the service quality violates the SLAs. Both the underestimation of provisioning and the overestimation of de-provisioning lead to penalties [3]. Thus, monitoring approaches and tools are critical in assessing the quality of services and providing an SLA violation report that helps both customers and providers. These approaches should support the understanding of services' actual behavior to improve or adjust the service operation to attain customer satisfaction and avoid possible penalties.

Various studies have analyzed the limitations of existing cloud monitoring approaches [4], [5]. In particular, these limitations are the fact that supported SLAs lack expressiveness with which to model real-world scenarios, the monitoring configuration is highly coupled with a given SLA specification, and the SLA violation reports provided

The associate editor coordinating the review of this manuscript and approving it for publication was Claudia Raibulet.

are difficult to understand [6]. Moreover, most commercial monitoring tools are tightly integrated with the cloud providers existing tools. For example, CloudWatch, offered by Amazon, is a monitoring tool that enables consumers to monitor their applications residing on AWS EC2 (CPU) service, but this tool does not have the ability to monitor an application component that may reside on other cloud provider's infrastructure such as GoGrid and Azure [4]. Thus, there is a need for providing open platform-independent monitoring tools, as well as uniform monitoring interfaces for different cloud providers [7].

Besides, if a given non-functional requirement (NFR) expressed in the SLA needs to be changed (e.g., owing to an SLA renegotiation), this can lead to significant changes in the monitoring infrastructure. Finally, another shortcoming in current solutions is the difficulty of using low-level metrics (e.g., latency, uptime) to define high-level indicators such as performance or availability [8]. This is in line with the results of a recent industrial study that revealed that cloud services monitoring is done with crude technology, mostly MySQL querying or similar (e.g., Nagios) [9].

We have addressed these issues by introducing a method for Monitoring cloud Services at RunTime (MoS@RT) [10]. We have also proposed a preliminary version of an infrastructure [11] that uses models at runtime (models@runtime) to allow the addition or modification of NFRs to be monitored and the selection of appropriate metric operationalizations depending on the actual cloud services capabilities without interruption to the services being executed. This model at runtime is causally connected to the running system, meaning that a change in the runtime model triggers a corresponding change in the running system and/or vice versa [12]. The infrastructure also provides users with facilities that enable them to configure the instructions to be applied during monitoring services (e.g., metrics, data sources).

This paper presents an enhancement and extension of the monitoring infrastructure that supports the proposed monitoring method and the empirical evaluation of the proposed method and infrastructure for monitoring cloud services in a specific cloud platform. In particular, the main contributions of this paper can be summarized as follows:

- The extension of the previously proposed infrastructure with additional data collection mechanisms to improve the infrastructure's interoperability by using third-party monitoring tools and a dashboard that allows users to visualize the results of QoS monitoring. Note that a preliminary version of the middleware component of the monitoring infrastructure was presented in [13].
- The design of a platform-specific architecture for instantiating the proposed infrastructure in Microsoft Azure.
- An evaluation method for predicting the likelihood of acceptance of MoS@RT.
- The design and execution of an experiment that uses the proposed evaluation method for assessing MoS@RT when performing the monitoring configuration of a cloud service in Azure.

Keeping in mind the high impact of user perceptions in the acceptance of a new solution, we propose an evaluation method that uses the Method Evaluation Model (MEM) [14] to evaluate the likelihood of acceptance of MoS@RT. MEM is based on the Technology Acceptance Model (TAM) [15], which analyses the perceived ease of use, perceived usefulness, and intention to use of participants by applying a method to predict its acceptance. Finally, we reported the results of an experiment where novice cloud software engineers used the monitoring configurator prototype of MoS@RT (a component of the monitoring infrastructure that requires user interaction) to configure the monitoring of a set of QoS requirements.

This paper is organized as follows. Firstly, related work on cloud services monitoring and empirical studies of monitoring tools and methods are discussed in Section 2. The monitoring infrastructure and its main components are introduced in Section 3. The instantiation of the monitoring infrastructure in Microsoft Azure is introduced in Section 4. The proposed evaluation method is described in Section 5, while the experiment, its planning, and execution are described in Section 6. The results are reported, analyzed, and interpreted in Section 7. This section also discusses the threats that might affect the validity of our results. Finally, our conclusions and final remarks are presented in Section 8.

II. RELATED WORK

Several methods and tools with which to support the activities of service monitoring have been proposed and reported over the last years [2], [4], [5], [15]–[18]. However, only a few studies empirically validate the existing methods or the new ones being proposed to the best of our knowledge. Empirical studies focus on establishing taxonomies to classify methods [4], [17]. This section discusses related works that report monitoring approaches and empirical studies that assess the usefulness of methods and tools for cloud service monitoring and SLA compliance verification.

A. METHODS AND TOOLS FOR MONITORING CLOUD SERVICES

Researchers have focused their efforts on developing solutions that can help cloud providers to track the SLA violations of certain service quality requirements. They have also focused on reporting the current state of monitoring solutions.

These solutions have been summarized in some secondary studies [4], [18]. In particular, Fatema *et al.* [4] have surveyed a range of monitoring tools that are currently in use to gain insight into their technical capabilities. The authors identify the desired capabilities of monitoring tools used to serve different cloud operational areas from the perspective of both providers and customers and present a taxonomy, including these capabilities. Therefore, these authors discuss the importance of monitoring techniques oriented to cloud Computing's service model given its multi-tenant nature by showing the important role and the need to develop systematic ways to monitor by taking into account specific

cloud characteristics. The study by Syed *et al.* [18] review the state-of-the-art related to cloud monitoring solutions for private and public clouds. The authors conclude that there is a need to carry out more research on cloud-native monitoring. It is also necessary to propose flexible, intelligent monitoring solutions, and user side SLA management.

Furthermore, many cloud providers enable their consumers to monitor cloud services using available monitoring tools for CPU, storage, and network [19]. These tools are often closely integrated with their platforms. For example, CloudWatch (offered by Amazon) is a monitoring tool that enables consumers to manage and monitor their applications that reside in AWS EC2 (CPU) services. However, this monitoring tool does not have the ability to monitor a service component that may reside in the infrastructure of other cloud providers such as GoGrid and Azure. Moreover, certain tools are unable to monitor QoS attributes and SLA requirements at the application level (e.g., security, elasticity, performance) because they mostly focus on monitoring hardware resources (CPU, storage, and networks). Besides, most commercial tools (e.g., CloudWatch, LogicMonitor) are not sufficiently flexible to allow a service provider to extend the QoS attributes provided to be monitored to assess SLAs' fulfillment. Other solutions are focused on providing cloud service monitoring [6], [16], [20], [21]. Keller and Ludwig [20] describe a framework named WSLA that can be used to specify and monitor SLAs for web services and have developed a prototype of a WSLA compliance monitoring tool. However, they do not consider how to deal with different metric operationalizations to provide their measurement service with flexibility. Emeakaroha *et al.* [16] propose an application monitoring architecture named Cloud Application SLA Violation Detection architecture (CASViD). This architecture monitors and detects SLA violations at the application layer and includes resource allocation and deployment tools. However, this approach does not have a flexible means to change the requirements and metrics to allow them to be monitored at runtime.

Shao *et al.* [21] proposed a runtime model for cloud monitoring (RMCM), which denotes a running cloud representation by focusing on common monitoring concerns. Still, they do not mention specific non-functional characteristics for SaaS cloud environments and their metrics (e.g., scalability, availability, elasticity). Furthermore, they do not provide an SLA violation report and leave addressing NFRs from SLA as future work. Finally, Muller *et al.* [6] have designed and implemented SALMonADA, a service-based system for monitoring and analyzing SLAs to explain violations. Nevertheless, these authors do not help stakeholders to select alternative metric operationalizations at runtime depending on the platform, and the system requires advanced users with knowledge of metrics and details about specific platforms.

Modi *et al.* [22] presented an ontology-based automatic cloud services monitoring and management approach. In this study, the monitoring is performed at a cloud broker level using SLA and ontology. An algorithm is also proposed for prediction-based service provisioning. The authors developed an SLA ontology model for the semantic description of the QoS parameters. When an SLA is violated, an alert is sent to both service providers and users, and the approach applies its prediction-based algorithm. This algorithm finds the virtual machine with less load and allocates the service to that virtual machine to avoid further SLA violations. The scope of this approach is limited to this scenario.

Shatnawi *et al.* [23] proposed an approach called CloudHealth to assess the health of cloud services. This solution supports three main activities (i.e., configuration, deployment, and operation) and allows assessing high-level monitoring goals using a mapping between quality characteristics, metrics, and probes. However, the approach is not flexible enough since the configuration and deployment steps need to be repeated when monitoring goals change.

Alhamazani *et al.* [24] proposed a framework for QoS monitoring and benchmarking of cloud applications distributed across cloud layers (*-aaS) and spread among multiple cloud providers. That proposal provides the ability to monitor and benchmark the QoS of individual application components such as databases and web servers distributed across a heterogeneous public or private cloud. Nevertheless, the framework employs an agent-based approach for monitoring the application components that collects and sends specific QoS values requested by a global manager. A specific monitoring agent is defined for each cloud provider, which is not flexible enough to include, remove, or change monitoring requirements at runtime.

Lu *et al.* [25] introduced a cloud monitoring system for cloud platforms (JTangCMS), which is composed of monitoring agents implemented as pluggable monitoring components. The proposed system covers collecting, delivering, and processing monitoring data collected from services deployed on the cloud. For data collection, pluggable monitoring components to collect runtime information from different entities are provided. For data delivery, a data dissemination framework is implemented to transfer the huge amount of runtime information. For data processing, a cloud action platform is implemented to support cloud management decision-making. However, the collecting data agents are specifically implemented to extract the low-level information available from each cloud platform. In order to access to different counters or APIs of external tools, the agents need to be redefined. The objective of the proposed system is to provide flexibility by allowing pluggable monitoring components and to reason about a cloud service's behavior in terms of sequences of events to perform decision-making about the overall cloud application.

In summary, despite the number of approaches proposed to monitor the QoS of cloud applications, there is still a need for approaches to monitor high-level non-functional requirements and environments that are flexible enough to allow adding or modifying monitoring requirements at runtime. These changes in monitoring requirements may be due to the

renegotiation of SLAs or the need to know the measurement values of certain quality attributes that were not of interest when the monitoring requirements were initially established.

To address these problems, we proposed a monitoring process that exploits models at runtime to monitor non-functional requirements of cloud services [10]. This study presented the conceptual idea behind the use of models at runtime and focused on describing the high-level steps of a monitoring process and the structure of the metamodel at runtime. Then, we proposed a monitoring infrastructure to support this process [11]. The infrastructure integrates two main components: a monitoring configurator and a monitoring & analysis middleware, and this study focused on the monitoring configurator component by describing how the Runtime Quality Model proposed in [10] can be integrated with other models (i.e., Monitoring Requirements Model, SaaS Quality Model) to support the monitoring of cloud services. Specifically, the contribution of [11] was to describe the meta-models that support these three models and the interaction among them. Finally, we proposed a platform-independent middleware [13] to support the monitoring & analysis middleware component of the infrastructure. This middleware provided practical evidence that non-functional requirements or metrics for measuring specific quality attributes can be easily changed at runtime by using models at runtime. The study also showed a first instantiation of the middleware in a specific cloud platform (i.e., Azure).

Although our previous studies proposed a preliminary version of the monitoring method, its supporting infrastructure and middleware (MoS@RT), the infrastructure still needs to be improved in several direcions: i) to support additional data collection mechanisms to improve the infrastructure's interoperability; ii) to improve the monitoring dashboard to enable users to easely and effectivelly visualize data combined from different sources as well as select different visualization mechanisms to present the monitoring results; iii) to support the use of historical data to make predictions; iv) to instantiate the platform-independent middleware in other cloud platforms, and v) to improve the platform-specific middleware defined for Azure [13] by defining an architecture that clearly describes how the middeware is structured and operates in practice, so that researchers or practitioners can use it in other contexts. Finally, there is a need of empirical studies that demonstrate the usefulness of the solution when users interact with the proposed monitoring infrastructure.

B. EMPIRICAL STUDIES OF CLOUD MONITORING METHODS

As a young technology, cloud computing and its monitoring tools still lack a broad consensus of appropriate evaluation criteria. It is desirable to have appropriate monitoring tools, which should have been evaluated to ensure their efficacy and utility. Various empirical studies are discussed below.

Bodenstaff *et al.* [26] proposed MoDe4SLA, which allows the management and monitoring of dependencies between services in a composition. They empirically validated their approach through an experiment with 34 participants. The authors evaluated usefulness by asking experts to manage simulated executions of service compositions using MoDe4SLA. However, they did not focus on monitoring the service quality but rather on the results, which lead to a good composition of services.

The SLA@SOI project proposed a framework for the management of services based on SLAs. They presented evaluations that demonstrated the applicability of SLA@SOI [6]. They also presented a case study concerning the application of the SLA@SOI framework to eGovernment domains. However, the case study was instead a proof-of-concept, and the results were focused on the entire framework without paying attention to the evaluation of the monitoring approach.

Emeakaroha *et al.* [16] presented CaSViD and evaluated their proposal with a proof-of-concept. They evaluated two aspects: (i) the ability of the architecture to monitor applications at runtime in order to detect SLA violations, and (ii) its capacity to determine the effective measurement interval for efficient monitoring automatically. The evaluation lacked rigor, and the authors did not focus on the users' perceptions when using their solution.

Finally, several works report experiences regarding the use of monitoring tools to evaluate efficiency, latency, performance, and other low-level NFRs [27], [28], [29].

Montes *et al.* [28] proposed cloud monitoring, GMonE (Global Monitoring systEm), a Cloud monitoring tool. They evaluated the performance, scalability, and overhead of GMonE using an experimental testbed. They tested their approach benefits in a large-scale cloud environment, including high performance, low overhead, scalability, and elasticity. Meng *et al.* [27] performed extensive experiments in an emulated cloud environment with a real-world system and network traces. The results show that their approach achieves significantly lower monitoring costs, higher scalability, and better multi-tenancy performance than others. However, their evaluation did not include real environments and users to provide feedback and contribute toward enhancing the approach with their perceptions.

The analysis of the studies above has allowed us to identify some limitations in the empirical evaluation of cloud monitoring solutions, such as (1) the low number of empirical studies assessing the users' experience of using the monitoring tool; (2) the lack of studies that analyze the interaction between the monitoring solution and its users for the definition of the quality characteristics to be monitored; and (3) the analysis of the likelihood of intention to use a given solution when users need to monitor their cloud services.

III. MONITORING INFRASTRUCTURE

The Monitoring Infrastructure supports the monitoring process defined in [10], and it allows: i) the specification and configuration of NFRs to be monitored; ii) interaction with cloud services in order to assess their quality at runtime; and, iii) the service status to be observed at runtime and the

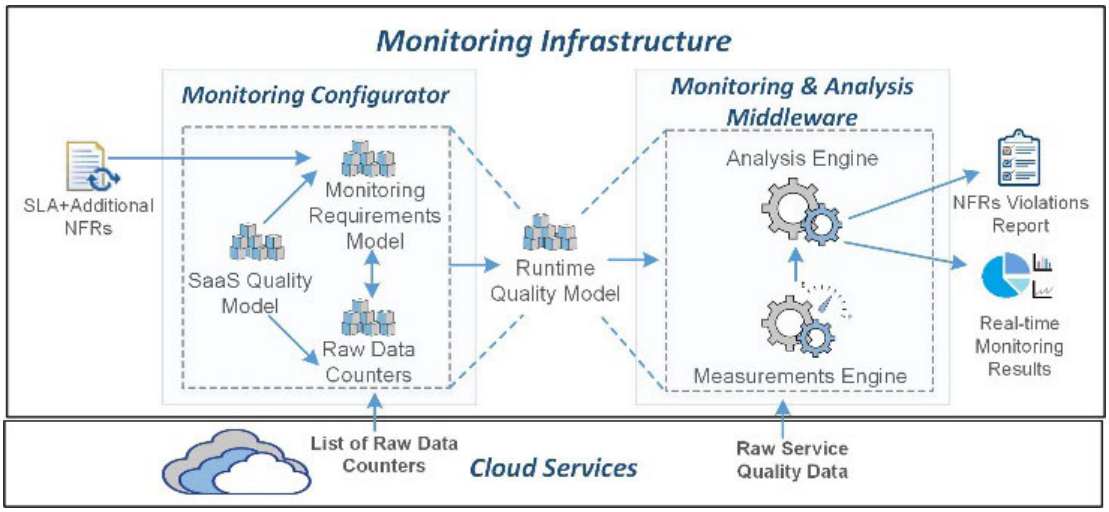

FIGURE 1. Monitoring infrastructure.

generation of reports containing any eventual SLA violation. Moreover, a set of components and artifacts that conform to the monitoring infrastructure has been defined to achieve these goals, as shown in Figure 1. The infrastructure uses models@runtime to provide the required degree of flexibility when defining metrics and different mechanisms to gather data from cloud services.

The monitoring infrastructure has two main components: the Monitoring Configurator and the Monitoring & Analysis Middleware. The Monitoring Configurator uses the Monitoring Requirements Model and the SaaS Quality Model (i.e., an ISO/IEC 25010-compliant quality model for cloud services built from a systematic literature review [30]) to configure the monitoring of services and obtain the Runtime Quality Model. The Monitoring & Analysis Middleware uses the Runtime Quality Model and relies on two engines: the Measurement Engine, which gathers raw data from services and applies the monitoring instructions, and the Analysis Engine, which compares the expected values with the monitored values and generates the SLA violations report.

In the following sections, we briefly describe each component of the monitoring infrastructure. These components and their meta-models have been presented in [11]. This infrastructure allows monitoring any NFR as long as raw data from the service to be monitored can be obtained, and a metric can be calculated from these data. However, the mechanisms for collecting data [11] allow gathering data from the running services in an interoperable and easy way. Therefore, it is possible to gather any kind of data even if the platform does not offer that information by using specific wrappers or by a third-party tool.

A. MONITORING CONFIGURATOR

Three models are used in the Monitoring Configuration (i.e., the Monitoring Requirements Model, the SaaS Quality Model, and the Runtime Quality Model). The Monitoring Requirements Model (see Figure 2(1)) contains all the NFRs that will be monitored. This model contains the SLA constraints, together with the corresponding thresholds, which should be evaluated. The SaaS Quality Model (see Figure 2(2)) represents SaaS quality characteristics' decomposition into measurable quality attributes and the different metric operationalization alternatives that can be used during the service monitoring process. A metric's operationalization consists of establishing a mapping between the metric's generic specification and the concepts represented in the software artifacts to be measured. The possibility of having several operationalizations allows the most appropriate measurement function to be selected (by considering the availability of raw data on a specific platform).

The Runtime Quality Model (see Figure2(3)), which is a model@runtime, specifies all the directives that are needed to access the services to be monitored during their execution. This model contains the actual parameters and instructions inherent to the platform, which can be retrieved using different methods (e.g., agents, APIs, platform tools, libraries). Figure 2(4) shows the Monitoring Configurator, which allows the definition of the Runtime Quality Model by mapping the metrics specified in the Monitoring Requirements Model with the platform-dependent formulas contained in the SaaS Quality Model. The monitoring configurator was implemented in ASP. NET using C# on the server-side to manage the three models in XML/XMI.

B. MONITORING AND ANALYSIS MIDDLEWARE

It has implemented the Monitoring & Analysis Middleware, shown in Figure 2(5), as a service to interact with the cloud services to be monitored. Moreover, it is possible to use different means to gather data from services deployed on any platform through wrappers and third-party solutions that allow the extension of a service to provide quality information. Here, it is shown how the platform-independent monitoring middleware can be instantiated.

We specifically show how the middleware design and implementation have been applied to monitor cloud

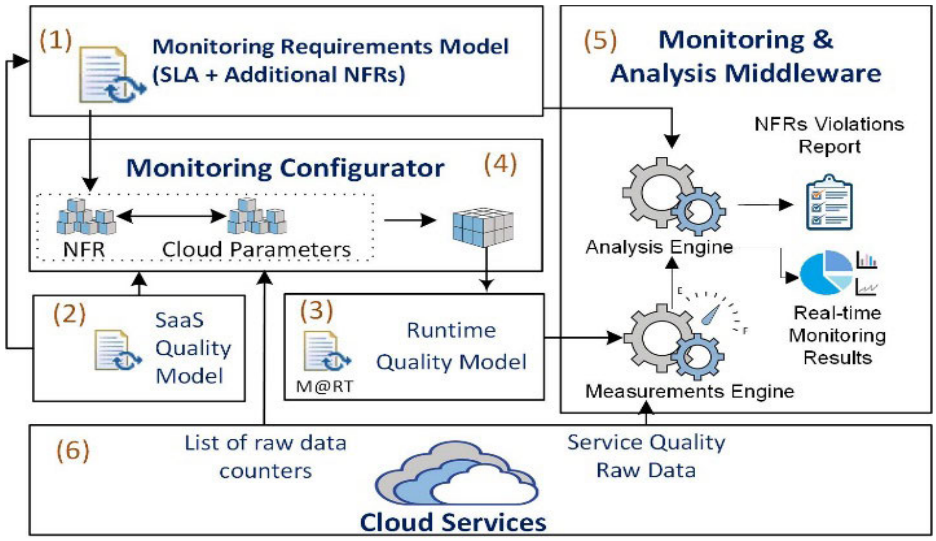

FIGURE 2. Monitoring configurator and monitoring & analysis middleware (MoS@RT).

services deployed on Microsoft Azure. However, similar actions can be taken to apply this solution to the monitoring of cloud services that are deployed on other platforms (i.e., Amazon Web Services, Google App Engine). In this approach, the middleware was deployed as an instance of the Worker Role on Microsoft Azure. Once executed, a Monitor object is created to search for the Runtime Quality Model to initiate the monitoring process. The Measurements Engine must gather data and calculate metrics regarding cloud services' behavior and performance, while the Analysis Engine determines whether the SLA and other NFRs for the services of interest are fulfilled.

Figure 3 shows the Measurement Engine with four possible data-gathering scenarios. The first two scenarios are supported by two data-gathering mechanisms (i.e., Platform Counters and Custom Counters) proposed as part of the previous version of our monitoring infrastructure [11].

The first scenario (1) shows the case in which raw data is gathered directly. This direct gathering of raw data can be achieved using the Microsoft Azure Diagnostics Service as the platform data retrieval mechanism. The data sources should be configured and directly retrieved from a set of Azure Performance Counters [31]. The Microsoft Azure Diagnostics has to be imported, and the data source in which the raw data will be stored and later manipulated should be set. In other platforms like AWS, performance counters can be obtained from the Performance Monitor Counter [31] in the same way as Azure Diagnostics.

The second scenario (2) is when there are no Performance Counters for direct use. Therefore, it is necessary to build Custom Performance Counters by combining Azure Performance Counters or other Custom Performance Counters. In this case, the Measurements Engine calculates Custom Performance Counters, and the Microsoft Azure Diagnostics Service can manage the result. In the Google app engine, the Cloud Monitoring API can be used to retrieve monitoring data and create custom metrics. This scenario represents the calculation of indirect metrics or Key Performance Indicators by using other direct metrics. It can be applied on any platform since monitoring data results from measurements calculated from data gathered using scenarios (1) and (3).

This paper extends the monitoring infrastructure with two new scenarios and additional data-gathering mechanisms to improve the infrastructure's flexibility and interoperability. The third scenario (3) is when data regarding the service quality cannot be obtained directly from the Azure platform or any other platform, and the corresponding cloud service has to be extended to provide the information needed. It can be done using wrappers that encapsulate the corresponding cloud service. The mechanism used to produce the data needed from the service is hardcoded in these wrappers, which can also be considered Custom Performance Counters, and store the data gathered in any storage solution. (e.g., Azure Storage). Finally, mechanisms that permit the use of third-party solution data are currently being developed as a fourth scenario, shown in Figure 3(3b), to provide interoperability with other solutions, which can be specialized to monitor certain NFRs or can provide useful data. When the monitoring directives have been included in the Runtime Quality Model, they are used by the Monitoring & Analysis Middleware.

The Monitoring & Analysis Middleware (see Figure 3) gathers raw data containing the quality information from the services using one of the scenarios detailed above. The raw data obtained is stored in an Azure Storage Account. Two tables are used to store the monitoring information: (1) the Direct Counter Table in which the raw data is gathered by the Diagnostics Service directly from the services; and (2) the Calculated Metrics Table, which contains the calculated metrics that are generated by the Measurements Engine and passed to the Analysis Engine (see Figure 3). It is also necessary to specify the sampling frequency of each Performance Counter, which may be different depending on the

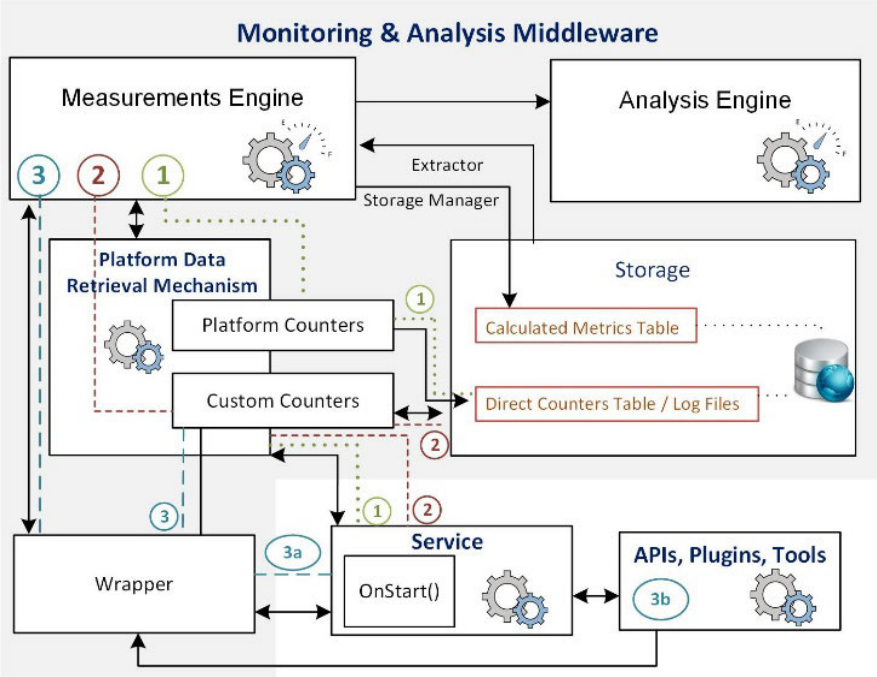

FIGURE 3. Platform-independent data extraction scenarios.

stakeholders' settings and the period in which the data are transferred to storage. Since the Runtime Quality Model can change, it is able to handle its updates at runtime, thus providing the monitoring infrastructure with flexibility. An Extractor class has been implemented in the Monitoring & Analysis Middleware to retrieve data from the tables so as to operate with it. Finally, the measurement engine provides the analysis engine with the monitoring data, and the analysis engine compares the measurements with the thresholds, assessing the quality of the service.

The main benefits of this Infrastructure are its ability to monitor application-specific quality requirements and the flexibility and maintainability of the Monitoring & Analysis Middleware. It occurs when new NFRs need to be added or modified or when a different metric operationalization is needed for a given quality attribute (e.g., using a more precise calculation formula to determine the probability of failure of a given cloud service). This advantage exists thanks to the Runtime Quality Model, which decouples those NFRs that will be evaluated and states how the calculation formulas from the Monitoring & Analysis Middleware will be applied.

IV. INSTANTIATING THE INFRASTRUCTURE

It has instantiated the infrastructure design in a specific cloud platform. Figure 4 shows the monitoring architecture structure, including its components and their dependencies with the cloud platforms. Then, each component has been labeled with an identification number. The SaaS Quality Model (1) contains the characteristics, sub-characteristics, attributes, and platform-independent metrics that can be used during the monitoring configuration. The Monitoring Requirements Model (2) contains the NFRs to be monitored. These NFRs have been obtained from the SLA and additional NFRs. This model is platform-independent.

The third component (3) is the Monitoring Configurator, which presents a front-end that guides users in configuring the monitoring requirements and obtaining the Runtime Quality Model. This component is implemented as part of the infrastructure for gathering the platform-specific parameters by using the libraries and tools provided by the same platform in which monitoring is performed. The Monitoring Configurator uses the SaaS Quality Model and the Monitoring Requirements Model as inputs and parses them. The parser is probably the most important effort in the platform-dependent implementation. However, it can be implemented only once by each platform and offered as libraries for future uses. The creation of multi-platform libraries can be considered as future work.

The Monitoring Configurator (3) allows the generation of the Runtime Quality Model (4), which contains the instructions to be applied during the monitoring activity. This model has the prescriptive and descriptive elements of a model@runtime. For more details about this model, please refer to [11].

The Monitoring and Analysis Middleware (5) uses the Runtime Quality Model and calculates the metrics to measure the requirements specified in the Monitoring Requirements Model. The Middleware gathers monitoring data by using several data extraction mechanisms. Since the monitoring middleware uses specific platform counters and needs to calculate certain values with platform instructions, it is considered platform dependent.

The data gathering mechanisms (6) allows retrieving information from different sources and feed the monitoring and

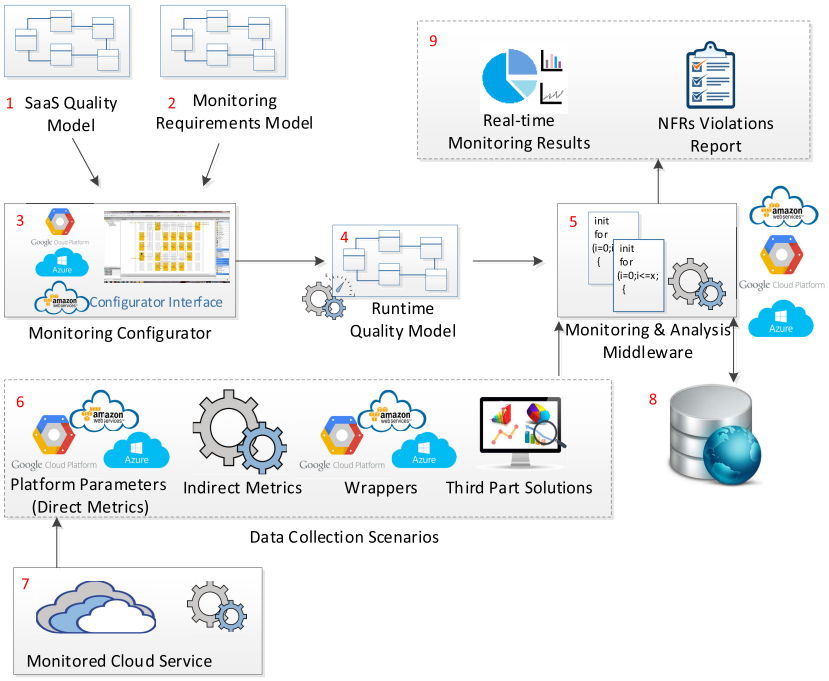

FIGURE 4. Monitoring architecture overview.

analysis middleware. These mechanisms access the cloud services (7) to gather the information from the services directly or perform some calculations (e.g., combining direct metrics for defining indirect metrics, using wrappers to call measurement functions, calling third-party tools). It can be seen as a virtual layer in which different mechanisms retrieve data. The database (8) stores the monitoring data gathered from the service (7) and the calculated metrics. Finally, the Monitoring & Analysis Middleware has been extended with a dashboard that allows users to visualize the monitoring results differently (9). They can be real-time monitoring charts and/or reports containing SLA violations.

A. INSTANTIATING THE MONITORING MIDDLEWARE IN MICROSOFT AZURE

The Monitoring and Analysis Middleware needs to be implemented in the same platform as the cloud services to be monitored due to instructions that need to be invocated to gather and manipulate data gathered from the services.

In a previous study [13], we presented a first instantiation of our platform-independent middleware in Azure and performed a case study on the monitoring of some cloud services. However, this platform-specific middleware should be applied to monitor other services, and more importantly, it should be improved based on the results of these case studies and the solution should be described in a high-level of abstraction so that researchers or practitioners can use it in other contexts. In the following, we describe the caracteristics of the refined middleware prototype, as well as the definition of an architecture that clearly describes how the middeware is structured and operates in practice.

The prototype has been developed to monitor services deployed in Microsoft Azure [32]. The ecosystem of the platform has determined the use of tools and language for building the prototype. Therefore, our prototype uses Visual Studio. NET [33] and C#. This language offers the advantage to gather and work with XML information and working with complex object models. Moreover, the middleware uses Azure Diagnostics [34] as a tool to gather data [13]. This tool allows gathering information from virtual machines and instances of these virtual machines, which are executed by a cloud service, and transfers these monitoring data to a storage account. In Azure, these data are modeled as class instances named Performance Counters [31]. A performance counter contains low-level metrics, and it is classified by using a name, which represents the source of information.

The monitoring middleware prototype was built to support different characteristics that MoS@RT should offer to its users: i) flexibility to support the monitoring of different types of NFRs and high-level QoS characteristics; ii) interoperability to allow the retrieving and manipulation of data gathered from different sources and now including a new mechanism to gather data from third parties (e.g., monitoring tools, APIs); iii) flexibility to change or define new monitoring requirements at runtime.

The middleware architecture for Microsoft Azure, shown in Figure 5, includes the following classes: (i) Monitor Class: this class gathers raw data and applies the metrics using the gathered data. It is the central and more important class; (ii) Metric: this class allows the representation of a measurable metric by using an operationalization; (iii) IOperationalization: it is an interface that defines the calculation methods the operationalizations. It is possible

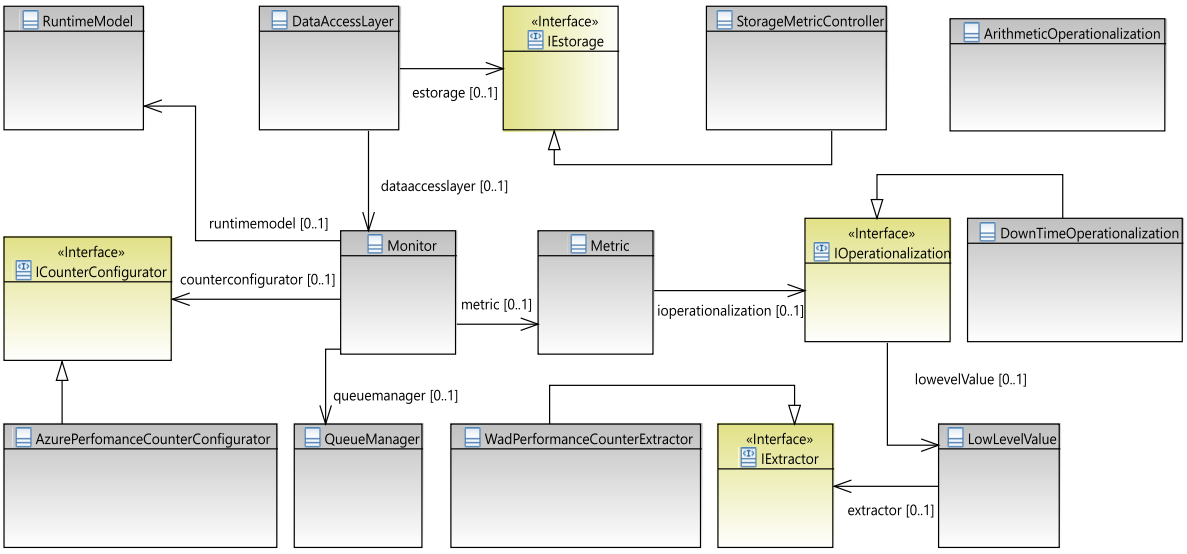

FIGURE 5. Middleware architecture for microsoft azure.

that certain metrics need a complex measurement function (e.g., downtime which needs to use conditionals and stores historical data to make comparisons); (iv) Arithmetic Operationalization: this class uses the NCalc library to make calculus defined by the operationalizations and send to the extractor the appropriate data to make these calculi; (v) DowntimeOperationalization: this class makes a custom measurement, it is necessary to define how data will be generated from particular needs of measurement.

As these values cannot be gathered directly from the performance counters of the platform, it can be considered as a second scenario where direct metrics are used to calculate indirect metrics; (vi) Low Level Value: this class represents raw data obtained from the service to be monitored; (vii) IExtractor: this is an interface which defines the gathering methods of raw data from different sources; (viii) WadPerformanceCounterExtractor: this class uses the platform counters of the service to be monitored to extract monitoring raw data; (ix) DataAccessLayer: this class allows the access to the data layer to store the monitoring and calculated data in a platform independent way; (x) Istorage: this is an interface that defines the access methods to the databases which stores the value of the metrics; (xi) StorageMetricsController: this is the implementation of the Istorage interface; (xii) Queue Manager: this class allows the access to an API in order to control the Runtime Quality Model; (xiii) ICounterConfigurator: this is an interface which allows the modeling of the methods for the dynamic configuration of the performance counters, if they are enabled for programming; (xiv) AzurePerformance CounterConfigurator: Uses the Performance Counters enabled in the Microsoft Azure Platform. This class activates and deactivates the data extraction depending on the needs stated by the Runtime Quality Model.

B. OPERATION OF THE MONITORING MIDDLEWARE IN MICROSOFT AZURE

This sub-section presents the operation of the monitoring middleware in Microsoft Azure. Firstly, the monitor is deployed and executed as a Worker Role instance of Microsoft Azure. Then, an object Monitor is created, which searches the Runtime Quality Model with which it is possible to initialize the service's monitoring. The Runtime Quality Model is loaded and managed by the Queue Manager class of Microsoft Azure.

When the Runtime Quality Model is loaded, this model is instantiated in a RuntimeModel object that allows accessing the configuration instructions. The operationalizations of the metrics (calculation formulas) show the performance counters that will be used to perform the measurements needed to evaluate the monitoring requirements. To do this, the monitoring middleware uses the appropriate ICounterConfigurator instance (e.g., AzurePerformance Counter Configurator) to activate the counters needed to monitor a given service. Then, the monitor begins to calculate the metrics contained in the Runtime Quality Model. The structure used to specify the calculation formulas (measurement functions) is compatible with the NCalc library [35], allowing parse any expression and evaluating the result, including static or dynamic parameters and custom functions. This solution allows the calculation of the metrics at runtime. Figure 6 shows an example of a metric operationalization in XML. A calculation formula may be composed of one or more operators and operands that need to be mapped to specific cloud platform parameters. This mapping is performed by an instance of IExtractor that in Microsoft Azure corresponds to the WADPerformanceCounterExtractor, which searches for the appropriate value for the operands. When the values for all the operands are obtained, NCalc calculates the metric value that will be stored in the database. The IStorage instance selected during the

```
<name>Defective Operations Per Million (DPM)</name>
<operationalization xsi:type="IndirectMetric">
  <name>DPM</name>
  <function>
    <formula>([36]-[35])/[36]</formula>
    <operands>
      <DirectMetric>
        <name>\ASP.NET Applications(*)\Requests Total</name>
        <identifier>[36]</identifier>
        <extractionRate>4</extractionRate>
        <extractionType>0</extractionType>
      </DirectMetric>
      <DirectMetric>
        <name>\ASP.NET Applications(*)\Requests Succeeded</name>
        <identifier>[35]</identifier>
        <extractionRate>4</extractionRate>
        <extractionType>0</extractionType>
      </DirectMetric>
      <DirectMetric>
        <name>\ASP.NET Applications(*)\Requests Total</name>
        <identifier>[36]</identifier>
        <extractionRate>4</extractionRate>
        <extractionType>0</extractionType>
      </DirectMetric>
    </operands>
  </function>
</operationalization>
```

FIGURE 6. Example of a metric operationalization in XML.

configuration indicates which database will be used to store the metrics. For this particular instantiation, it is used the table Calculated Metrics that has been created in the Azure Storage.

This process is done iteratively for each monitoring requirement. When there is a new requirement or a change in an existing requirement in the Runtime Quality Model (RuntimeModel), the Monitor reloads this model and re-calculate the modified calculation formulas. Therefore, the changes are applied at the configuration phase avoiding any change in implementing the monitoring infrastructure.

V. EVALUATION METHOD

The evaluation method proposed in this paper is based on the adaptation of the Method Evaluation Model (MEM) [36] for its use with MoS@RT. MEM provides a theoretical model and an associated measurement instrument for evaluating information systems (IS) design methods. It is based on the Technology Acceptance Model (TAM) that combines two different but related dimensions of method "success": actual effectiveness and practice adoption. Therefore, it results appropriate to evaluate the likelihood of acceptance and the actual impact of a cloud services monitoring methods in practice. The likelihood of acceptance is indicated for recently-proposed methods, while the actual impact can only be measured for well-established methods [37].

A. THE METHOD EVALUATION MODEL (MEM)

MEM's main contribution incorporates two different method success aspects: actual efficacy and actual usage (see Figure 7). It means that the adoption of a method in practice depends not only on whether it is actually effective (pragmatic success) [37] but also whether the users of the method perceive it to be effective (perceived success).

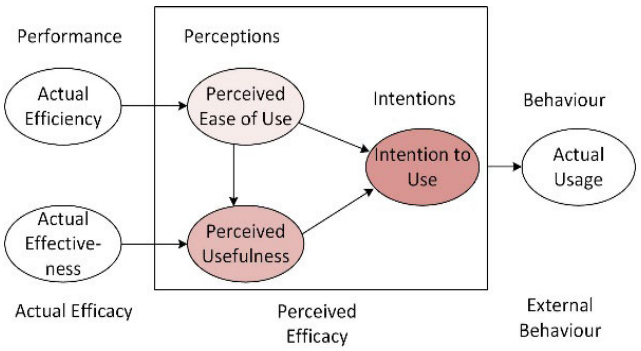

FIGURE 7. The method evaluation model (MEM).

Both aspects must be considered when evaluating cloud monitoring methods. Figure 7 shows the model's constructs, along with the hypothesized causal relationships among the model's constructs.

In the MEM, efficacy is defined as a separate construct, different from efficiency and effectiveness. The efficacy construct is derived from Rescher's notion of pragmatic success [38], which is defined as the efficiency and effectiveness with which a method achieves its objectives. Evaluating a method's efficacy requires measuring both the effort required (efficiency) and the results' quality (effectiveness).

The MEM constructs are based on the Technology Acceptance Model (TAM) [15], a well-known and thoroughly empirically validated model for information technologies evaluation. The constructs of the MEM are: actual efficacy and perceived efficacy.

Actual Efficacy has two performance-based variables: i) Actual Efficiency: the effort required to apply a method,

and ii) Actual Effectiveness: the degree to which a method achieves its objectives. This construct is related to the quality of the artifact(s) obtained by applying the method. According to Rescher [38], all methods are intended to achieve certain objectives. Rescher defines a method as ''a collection of rules and procedures designed to assist people in performing a particular task.'' Different objectives define different types of methods. It means that specific dependent variables will need to be defined for each class of methods to measure performance regarding its specific objectives.

Perceived Efficacy has two perception-based variables:

- **Perceived Ease of Use (PEOU):** the degree to which a person believes that using a particular method would be free of effort. PEOU represents a perceptual judgment of the effort required to learn and use a method.
- **Perceived Usefulness (PU):** the degree to which a person believes that using a particular method would enhance his/her job performance. This variable represents a perceptual judgment of the method's effectiveness. There is a causal relationship in the model that indicates that PU can be determined by PEOU.
- **Intention to Use (ITU):** the extent to which a person intends to use a particular method. This variable represents a perceptual judgment of the method's efficacy–its cost-effectiveness. This variable is used to predict the likelihood of a method being accepted in practice. The hypothesized causal relationships suggest that perceived ease of use and perceived usefulness directly affect intentions to use a method.
- **Actual Usage:** a behavior-based variable, defined as the extent to which a method is used in practice. It measures the actual impact of a method in practice. According to the hypothesized causal relationship, actual usage will be determined by the intention to use.

B. ADAPTING MEM TO EVALUATE MoS@RT

The first step involved in adapting MEM is to define the specific objectives of the method to be evaluated. The general constructs of MEM can then be instantiated into concrete dependent variables based on these objectives. We believe that cloud monitoring methods such as MoS@RT have three primary objectives:

- To support the configuration of the service monitoring. This objective involves several activities: the specification of NFRs to be monitored, the selection of appropriate quality attributes and metrics, and the specification of how raw data from the services will be obtained. Several approaches show that a monitoring configuration phase is necessary to specify how these activities will be performed (e.g., [8], [16], 39], [40]).
- To support the service monitoring. This objective involves gathering data, performing the measurements, and assessing the quality of services based on the monitoring configuration. These activities can be performed automatically by a monitoring engine (e.g., middleware, agents) which is used by the majority of the existing monitoring solutions (e.g., [4], [39], [41]). Data gathering is performed by means of performance counters, wrappers, third part APIs, and indirect metrics (see Figure 4 (6)).
- To report the monitoring results. This objective is concerned with the presentation of the monitoring results through graphic charts or SLA reports [13], [39], [42].

In this study, we focus on the first objective (i.e., monitoring configuration) using a particular monitoring method since it is related to essential tasks for monitoring cloud services. Also, these are the only tasks that require user intervention. Furthermore, the purpose of MoS@RT is to raise the level of abstraction in the definition of the service monitoring configuration and provide the monitoring infrastructure with flexibility to allow the monitoring requirements or the metric operationalizations of these requirements to be changed at runtime without the modification of the underlying infrastructure. We have accomplished these goals by using models at run-time. Therefore, the most critical activity to be evaluated is the monitoring configuration since it is related to the generation of the models at runtime. Note that once the configuration is defined indicating what is to be monitored and how (i.e., which metrics and measurement functions are to be used), the calculations can be performed automatically by a monitoring engine (e.g., the Monitoring & Analysis Middleware in the case of MoS@RT). This means that although the objectives 2 and 3 are relevant from a practical point of view, they are less relevant from a scientific point of view. Nevertheless, we validated that the models at runtime, generated by the participants, were correctly processed by the Monitoring & Analysis Middleware.

Evaluating the efficacy of MoS@RT involves measuring the effort required to apply the method (input) and the quality of the monitoring results (output). The effort required to understand and/or apply the method (i.e., actual efficiency) can be measured using several measures, such as time or cognitive effort. The quality of the method's result (actual effectiveness) can be measured by evaluating the monitoring results produced using the method (i.e., whether the monitoring configuration is correctly performed). The following performance-based variables are therefore used to measure the participants' actual efficiency and actual effectiveness as regards configuring the NFRs to be monitored:

- **Actual effectiveness:** the ratio between the number of NFRs correctly configured and the total number of NFRs to be configured.

$$Effectiveness = \frac{\#\ NFRs\ correctly\ configured}{Total\ \#\ NFRs\ to\ be\ configured} \quad (1)$$

- **Actual efficiency:** the time spent configuring the total number (n) of NFRs to be monitored.

$$Efficiency = \sum_{i=1}^{n} Time\ Configuring\ NFR_i \quad (2)$$

Note that the configuration of the NFRs to be monitored involves several tasks: selection of quality requirements, selection of quality attributes, selection of

platform-independent metrics, mapping the platform-independent metrics to platform-specific metrics (using specific measurement counters from the selected cloud platform) and generation of the runtime quality model that contains all the configuration directives. These tasks are further explained in Section VI(B).

In order to measure the perception-based variables, we relied on an existing measurement instrument for the MEM, although we adapted this instrument for use with MoS@RT (basically by rewording statements). Figure 8(a) shows how the MEM was operationalized to evaluate our proposed method in terms of its utility to configure the monitoring of cloud services. To measure each of the three main constructs (PEOU, PU, and ITU), we defined sets of questions based on the items shown in Table 1. Figure 8(b) shows the theoretical model proposed to evaluate the method. Basically, we use performance-based measures as influencing factors for perceived-based variables. According to this MEM adaptation, the likelihood of MoS@RT being accepted in practice can be predicted by testing the following hypotheses:

- $H1_0$: MoS@RT is perceived to be difficult to use to configure the monitoring of cloud services, $H1_1 = \neg H1_0$
- $H2_0$: MoS@RTis not perceived to be useful to configure the monitoring of cloud services, $H2_1 = \neg H2_0$
- $H3_0$: There is no intention to use MoS@RT to configure the monitoring of cloud services in the future, $H3_1 = \neg H3_0$

These hypotheses relate to a direct relationship between using this monitoring method and the users' performance, perceptions, and intentions. The evaluation model also proposes a set of hypotheses that indicate causal links between dependent variables (such as performance affecting perceptions or perceptions influencing intentions). These hypotheses are meant to validate the structural part of the MEM:

- $H4_0$: PEOU will not be determined by efficiency. $H4_1 = \neg H4_0$: The rationale for this hypothesis is that efficiency represents a performance-based measure of actual efficiency, while PEOU represents a perception-based measure of efficiency. According to MEM, efficiency measures the effort required to apply the method, which should determine perceptions of the effort required.
- $H5_0$: PU will not be determined by effectiveness. $H51 = \neg H5_0$: The rationale for this hypothesis is that effectiveness represents a performance-based measure, while PU represents a perception-based measure of effectiveness. According to MEM, perceptions of effectiveness should be determined by actual effectiveness.
- $H6_0$: PU is not determined by PEOU. $H6_1 = \neg H6_0$: This hypothesis is taken from the TAM, in which PEOU was found to have a direct influence on PU.
- $H7_0$: ITU is not determined by PEOU. $H7_1 = \neg H7_0$: This hypothesis is taken from the TAM, in which PEOU was found to influence ITU.
- $H8_0$: ITU is not determined by PU. $H8_1 = \neg H8_0$: This hypothesis is taken from the TAM, in which PU was found to have a direct influence on ITU.

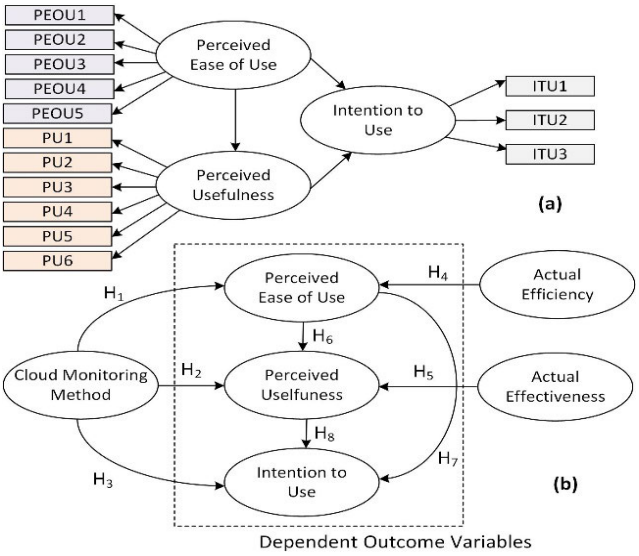

FIGURE 8. (a) Survey instrument and (b) Theoretical model.

The evaluation model consequently denotes that the acceptance of MoS@RT in practice can be predicted on the basis of particpants' perceptions of their ease of use and usefulness. Moreover, it is important to examine and measure the correlation between intention to use and actual usage when employing models based on TAM. However, the actual usage measures actual impact in practice (as opposed to potential impact, defined by Intention to Use). It may be evaluated by employing surveys of practice (it cannot be used to evaluate newly-proposed methods but only in assessing established ones). Table 1 shows the items defined to measure the perception-based variables. These items were combined in a survey with 14 questions.

The survey was used to gather the participants' perceptions with regard the utility of MoS@RT to configure the monitoring of cloud services. The items were formulated using a 5-point Likert scale, with the opposing-statement question format. Various items within the same construct group were randomized to prevent systemic response bias. PEOU is measured using five items in the survey. PU is measured using six items in the survey. Finally, ITU is measured using four items in the survey. Moreover, to ensure the balance of items, approximately half the questions were negated to avoid monotonous responses. The measurement instrument can be accessed at: http://goo.gl/forms/JxMEh4TY5t.

VI. EXPERIMENT DESIGN

As there is currently no standard or widely accepted cloud monitoring method, it is not possible to evaluate MoS@RT against a control method. Therefore, it has been decided to carry out a quasi-experiment to evaluate MoS@RT and test the evaluation method proposed in Section 5 empirically.

A quasi-experiment is an empirical inquiry, similar to an experiment, in which the assignment of treatments to subjects

TABLE 1. Survey for measuring the perception-based variables.

Question	Positive Statement (5 Points)
PEOU1	The method for monitoring cloud service quality is complex and difficult to follow.
PEOU2	In general, the method for monitoring cloud service quality is easy to understand.
PEOU3	The steps used to configure the monitoring of cloud service quality are clear and easy to understand.
PEOU4	The method for monitoring cloud service quality is easy to learn.
PEOU5	I believe that it would be easy to become skilled in using this method.
PU1	I believe that this method would reduce the time and effort required to monitor cloud services.
PU2	In general, I consider that the method for monitoring cloud service quality is useful.
PU3	I believe that the process of setting up non-functional requirements of this method is useful for monitoring cloud service quality.
PU4	I believe that the method is sufficiently expressive regarding defining how the measurement of NFRs should be performed.
PU5	The use of this method would enable me to improve my performance when monitoring cloud service quality.
PU6	In general, I think that this method will allow me to monitor the cloud service quality adequately.
ITU1	If I had to use a method to monitor cloud service quality in the future, I would consider using this method.
ITU2	If it is necessary, I will use this method in the future.
ITU3	I would recommend the use of this method for monitoring cloud service quality.

cannot be based on randomization but emerges from the characteristics of the subjects or objects themselves [43].

The quasi-experiment was designed according to the experimental process proposed by Wohlin *et al.* [43]. Its main goal was to empirically evaluate the perceived efficacy of MoS@RT when used by a group of users to configure the monitoring of a cloud service in the Azure platform. To this end, the monitoring method and the underlying infrastructure were applied to predict the likelihood of acceptance of the monitoring configuration activity as part of the method in practice (i.e., the monitoring configuration task described in Section 3.1.). We believe that a perception-based study can help us understand users' needs to refine the monitoring configuration activity, which is the only part of the method that involves human intervention.

A. EXPERIMENT GOAL

According to the Goal-Question Metric (GQM) paradigm [8], the goal of the experiment is defined as follows: *Analyze* the monitoring configuration specifications obtained with MoS@RT *for the purpose of* assessing them *with respect to* the effectiveness, efficiency, perceived ease of use, perceived usefulness and intention to use of participants when applying the method to configure the monitoring of a cloud service *from the point of view of* novice cloud software engineers *in the context of* third-year Computer Science undergraduates at the Universitat Politècnica de València.

The research questions addressed by the experiment are:

- RQ1: Is MoS@RT perceived to be both easy to use and useful to configure the monitoring of cloud services? If so, are the users' perceptions resulting from their performance when configuring the quality requirements to be monitored?
- RQ2: Is there an intention to use MoS@RT to configure the monitoring of cloud services in the future? If so, is the intention to use it a result of the perceptions experienced by the participants when configuring the quality requirements to be monitored?

These research questions were evaluated by testing a number of hypotheses (see Figure 6(b)). The following hypotheses addressed the first research question:

- $H1_0$: MoS@RT is perceived to be difficult to use to configure the monitoring of cloud services. $H1_1 = \neg H1_0$
- $H2_0$: MoS@RT is not perceived to be useful to configure the monitoring of cloud services. H21= ¬H20
- $H4_0$: Perceived Ease of Use is not determined by Efficiency. $H4_0 = \neg H4_1$
- $H5_0$: Perceived Usefulness is not determined by Effectiveness. $H5_0 = \neg H5_1$

The second research question was addressed through the formulation of the following hypotheses:

- $H3_0$: There is no intention to use MoS@RT to configure the monitoring of cloud services in the future. $H3_0 = \neg H3_1$
- H60: Perceived Usefulness is not determined by Perceived Ease of Use. $H6_0 = \neg H6_1$
- $H7_0$: Intention to Use is not determined by Perceived Ease of Use. $H7_0 = \neg H7_1$
- H80: Intention to Use is not determined by Perceive Usefulness. $H8_0 = \neg H8_1$.

B. EXPERIMENT PLANNING

This section describes all the activities performed when planning the experiment.

1) CONTEXT SELECTION

The context is determined by the monitoring method to be evaluated, the cloud service selection to be evaluated, and the selection of participants.

The monitoring method to be evaluated is the cloud Monitoring Services at RunTime (MoS@RT) method. Here, the focus is on the Monitoring Configuration activity, which is used to generate the Runtime Quality Model. This has been considered as a primary activity because of the need for user interaction with the monitoring infrastructure. The participants, therefore, performed the Monitoring Configurator role (see Figure 9), which includes the following tasks: (i) Quality Attribute Selection, (ii) Measure Selection, (iii) Metric Mapping, and (iv) Monitoring Model Generation. These subjects should preferably possess technical expertise on Cloud Computing platforms and knowledge about software metrics. This is necessary in order to understand the monitoring configuration activity and to be able to properly select the quality attributes and metrics from the SaaS Quality Model and establish the mappings of the metrics operands with the counters available in a specific cloud platform.

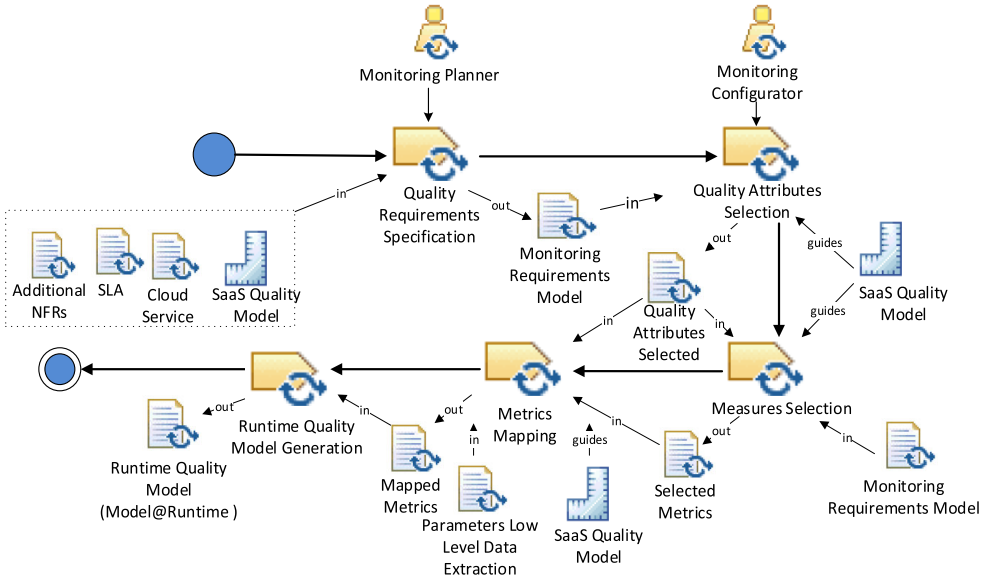

FIGURE 9. Monitoring configuration task.

It is assumed that the Monitoring Configurator would use the Monitoring Requirements Model, provided by the Monitoring Planner (see Figure 9), which includes the NFRs contained in the SLA and the additional NFRs to be monitored. The participants were also provided with the characteristics, sub-characteristics, quality attributes, and metrics included in the SaaS Quality Model so they could select the quality attributes and metrics needed to evaluate the NFRs specified in the SLA. The participants specifically applied the Monitoring Configuration tasks shown in Figure 9 to monitor one of the services related to a specific domain. When the monitoring configuration was completed, the participants generated the Runtime Quality Model used by the Monitoring & Analysis Middleware (see Figure 1) to monitor the service's quality automatically.

The cloud service to be evaluated belongs to an online auction site. An online auction is a process of buying and selling goods or services by offering them up for bid and then selling the item to the most appropriate bidder through the Internet. These sites allow thousands of users to make bids and should ensure high levels of reliability, availability, elasticity, and accuracy. Auction sites allow bids, which come in many different formats, but the most popular are ascending and descending bids. The service to be evaluated in this experiment is related to a site for ascending bids. These sites have proliferated on the Internet and represent a clear example of what we wish to show concerning the necessary quality requirements that can be monitored using our approach. The process is the following: (1) Online site users should sign-up and buy credits; (2) A user interested in a particular product uses credits to place a bid; (3) A user waits for the timer to reach 0 (there is a clock, which counts down to zero); (4) If there is another user who is also interested in the product and wishes to buy it, bids and the time restarts; finally, (5) If no one else bids at the auction, it ends, and the product is awarded to the last user to bid, who is the winner.

Many sites (e.g., MadBid, QuiBids, DealDah) specialize mainly in bids for electronic products, jewelry, home products, cars, among others. Here, a set of cloud services products is offered to provide these site owners with the NFRs needed for this domain.

Of the services that these sites offer (e.g., Inventory Service, Auction Service, Order Bids Service, User Service, Incident Handling Service, and Payment Service), we have chosen the Auction Service, which is the most important and critical service. The high level of quality that it demands makes this experiment an interesting problem to be solved.

Specifically, the monitoring configuration consists of three NFRs to be monitored: Reliability, Availability and Latency. The Monitoring Requirements Model is provided to the participants who plays the Monitoring Planner role. This model specifies the NFRs and the metrics that should be used as well as the thresholds to be considered. For example, the reliability of the service will be measured by the number of defective operations per million using (3) and the service should have a maximum of 10 defective operations per million, representing a reliability of 99.999%.

$$DPM = \frac{Operations\ Attempted - Operations\ Successful}{Operations\ Attempted} \tag{3}$$

Finally, 58 participants were selected. All of them were Computer Science undergraduates at the Universitat Politècnica de València. We took a convenience sample consisting of two groups of students attending a course on Software Quality in the Autumn of 2018 (the morning group consisted of 37 participants and the afternoon group of 21 participants). These students have been selected because of their strong knowledge of quality models and metrics

(they defined quality models as part of the course). They received training in cloud computing platforms and cloud monitoring, and the experiment was organized as a compulsory part of the course.

2) EXPERIMENTAL TASKS

The quasi-experiment consisted of three tasks.

- **Task 1:** The categorization of three NFRs to be monitored (i.e., reliability, availability and latency) and the selection of metrics and operationalizations. For each NFR, the participants had to use the SaaS Quality Model to select the appropriate quality characteristic, attribute, and platform-independent operationalization of a metric, which would allow them to evaluate that particular NFR. The metric selected had to be equivalent to the metric expressed in the Monitoring Requirements Model, which describes the NFRs to be monitored. The metrics and operationalizations involved in this task were platform-independent, and a prototype of the monitoring configurator supported the task.
- **Task 2:** The selection of the most appropriate platform-dependent formula for each metric selected in Task 1. It allowed the mapping to be made between the generic definition of the metric (i.e., measurement function) and the platform's low-level parameter counters, allowing raw data to be gathered from the service and their monitoring at runtime. Here, a list of platform counters from the Azure platform was displayed on the configurator, consequently allowing the participants to build the calculation formula (measurement function) with which to evaluate each NFR to be monitored. For example, when building the platform-specific metric for the formula in (3), *OperationsAttempted* is substituted by *\ASP.NET Applications(*)\Requests Total* which is the specific counter from the Azure platform that allows measuring this concept.
- **Task 3:** The modification of an NFR owing to an SLA renegotiation. The participants had to analyze the modified NFR to determine the changes in categorizing the attributes and metrics (Task 1) and the needed platform-dependent metrics and low-level parameter counters from the platform (Task 2).

3) VARIABLES

Table 2 shows the perception-based dependent variables of interest in the study, which were used to evaluate MoS@RT in practice.

These variables were measured using a Likert scale questionnaire with a set of 14 closed questions (i.e., 5 for PEOU, 6 for PU, and 3 for the ITU), as shown in Table 1. The closed questions were formulated by using a 5-point Likert scale. The aggregated value of each subjective variable was calculated as the arithmetical mean of the answers to the questions associated with each subjective, dependent variable. Table 3 shows the performance-based variables of interest (see Section 5.2) and the formula used to determine their values.

TABLE 2. Perception-based dependent variables.

Variable	Description
PEOU	The degree to which participants believe that learning and using MoS@RT will be effort-free.
PU	The degree to which participants believe that using MoS@RT will increase their performance.
ITU	The extent to which participants intend to use MoS@RT. It represents a perceptual judgment of the method's efficacy and can be used to predict the acceptance of the method in practice.

TABLE 3. Performance-based dependent variables.

Variable	Description
Effectiveness	$\frac{\sum_{i=1}^{n} Correct\ performed\ task_i}{n}$
Efficiency	$\sum_{i=1}^{n} Time\ executing\ task_i$

As mentioned previously, the experiment consisted of three tasks. In the first and second tasks, the participants configured the monitoring of three NFRs. Each NFR was regarded as one-third of the total value of each task. Therefore, the first and second task's effectiveness is the sum of the corrected performed actions with each NFR. The *effectiveness* of the third task can range from 0-1, signifying that the total *effectiveness* is the sum of the corrected tasks performed divided by the total number of tasks *(i.e., three)*. Finally, as the MEM [36] specifies, *efficiency* was measured as the total time spent configuring each NFR in each task performed by the participants.

4) INSTRUMENTATION

Multiple documents were defined as instrumentation for the quasi-experiment.[1] The documentation included: (i) a booklet that contained the Auction Site's description with the service to be monitored, the NFRs to be monitored, and the three tasks to be performed by the participants. The subjects were asked to write down the time before starting to solve the tasks; (ii) a detailed annex as support, which described each NFR to be monitored; (iii) the SaaS Quality Model; (iv) a list of parameter counters provided by the Azure platform; (v) a guide to MoS@RT to be used during the experiment as reference material; (vi) the monitoring configurator; and (vii) the survey, which contained both the closed questions in order to analyze the subjective variables and the open questions to enable the subjects to express their opinion about the method and the supporting infrastructure.

Configuration data were collected by using the monitoring configurator. The time spent on each task, and the data obtained from Task 3 were collected using the booklet described above. Finally, the questionnaire data were collected online.

[1]The experiment material is available at: http://goo.gl/tiFIU2

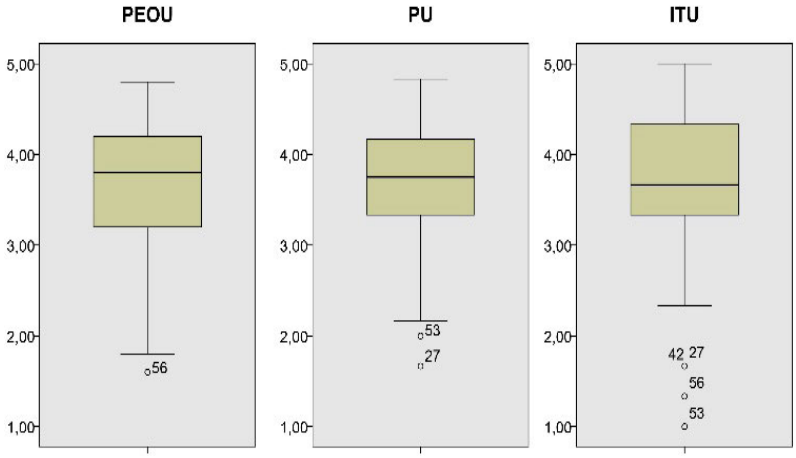

FIGURE 10. Box-plots for PEOU, PU, and ITU variables.

C. OPERATION AND EXECUTION

Three training sessions of 120 minutes each were performed before the experimental session to present cloud computing and monitoring concepts and train the participants in the use of MoS@RT. The training included the use of the Monitoring Configurator from the Monitoring & Analysis Middleware and the tasks involved in the configuration process. As a training example, we used the Open Reference Case (ORC) proposed in [41], which was used as an open-source demonstrator to highlight the achievements of the EU project SLA@SOI. The ORC is an extension of the CoCoMe implementation [44], which provides a service-oriented retail solution that can be used in a supermarket trading system to handle the sales and stocking process. The set of services was deployed as SaaS on the Microsoft Azure platform. During the training, the participants used the Monitoring Configurator to configure the monitoring of two NFRs (i.e., efficiency and service accuracy) for the Inventory service.

After the training sessions, the execution of the quasi-experiment took place. The experiment execution was controlled, meaning that no interactions took place between the participants. The experiment was conducted in two sessions (i.e., one session with the morning group and another with the afternoon group). Each session had an expected duration of 90 minutes. However, the participants were allowed to finish the experiment even when this time was up to mitigate a possible ceiling effect. The experimenters clarified any questions that arose during the experimental sessions. After the experimental tasks, the participants were asked to fill in the post-experiment survey questionnaire shown in Table 1.

Finally, the experimenters analyzed the definition of the models at runtime generated by the participants and performed some tests to ensure the validity of the configurations generated by the participants. The tests consisted in using the generated models at runtime as input to the Monitoring & Analysis middleware. Since all the generated models could be read and processed correctly by the monitoring infrastructure, they were all considered as valid.

VII. ANALYSIS AND INTERPRETATION

We used statistical tests, descriptive statistics, and boxplots to analyze the data collected. Since the two groups' participants had the same profile (see Section 6.2.1), we combined the data into one group. The data were analyzed according to the hypotheses stated. The results were obtained by using SPSS v20 with an $\alpha = 0.05$.

A. ANALYSIS OF USER PERCEPTIONS

Figure 10 shows the boxplots for each the PEOU, PU, and ITU variables in which we can see that the mean for each variable is higher than the Likert neutral value (since our variables are calculated as the mean of 5 points Likert-scale questions, the neutral value is 3).

The boxplots show some abnormal data-points (i.e., participant_id = 27, 42, 53 y 56), which correspond to participants that did not attend the training sessions. There, participants were removed from the subsequent analysis since they had not followed the experimental protocol. Once the aforementioned data-points had been removed, the Shapiro-Wilk test was applied to check whether the data were normally distributed to select which test would be used to check hypotheses H1, H2, and H3.

Table 4 shows the Shapiro-Wilk test results for the variables being studied. Statistical tests were applied to verify the hypotheses by comparing whether the mean of the responses to the questions related to a given variable was significantly higher than the Likert neutral value. For the variables PU and ITU, which have a normal distribution ($p > 0.05$), the hypotheses were tested by applying the one-tailed parametric t-test. In contrast, the variable PEOU, which does not have a normal distribution ($p < 0.05$), was tested by applying the one-tailed one-sample Wilcoxon test with a test value equal to three the Likert neutral score in the questionnaire. The test results allowed us to reject the null hypotheses $H1_0$, $H2_0$, and $H3_0$, signifying that the participants perceived MoS@RT to be easy to use and useful, and they also expressed their intention to use this method if they have to monitor cloud services in the future. These results are

TABLE 4. Shapiro-wilk test for the perception-based variables.

Var	Min	Max	Mean	Std. Dev.	Std. E.	1-T. p-value	Shapiro-Wilk test p-value
PEOU	2.00	4.80	3.710	0.6978	0.09496	<0.001**	0.012*
PU	2.83	4.83	3.833	0.5005	0.06811	<0.001	0.302
ITU	2.33	5.00	3.809	0.6364	0.08661	<0.001	0.102

*The variable does not approach a normal distribution
** Results of the one-tailed one-sample Wilcoxon test

supported by several positive comments that the participants gave in the open questions of the questionnaire: (e.g., "*I never monitored cloud services before but this method provided me useful guidelines on how to perform the monitoring configuration of cloud services,*" "*I found the method easy to use – although more information about the meaning of the performance counters from the Azure platform could be provided,*" "*I found the method really useful. I especially liked the SaaS Quality Model and the information about quality attributes and metrics that it provides*".

B. ANALYSIS OF USER PERFORMANCE

The participants' effectiveness and efficiency were measured when applying MoS@RT in practice. Table 5 presents the descriptive statistics for the performance-based variables. Note that the outliers identified previously have been eliminated from this analysis. The total effectiveness was, on average, 91.26%, indicating that almost all the participants were able to perform the monitoring configuration for a set of quality requirements correctly.

TABLE 5. Descriptive statistics for the performance-based variables.

Variable	Min	Max	Mean	Std. Dev.
Effectiveness	0.33	1.00	0.9126	0.1454
Efficiency	14.00	56.00	26.6379	8.57

The efficiency was calculated as the effort required (in minutes) to apply a method [36]. The results show that the participants' efficiency ranged from 14 to 56 minutes. We are aware that people's efficiency may vary considerably when configuring a service monitorization. It depends on many factors such as experience, the quality of the specifications, and tools. Notwithstanding these limitations, the purpose here was to collect this data to test whether the participants' perceptions were driven by their performance. These results also provided us with some bases to understand the performance of people using the method.

C. CAUSAL RELATIONSHIPS ANALYSIS

This section aims to validate the adapted evaluation method. This has been done by testing the structural part of the proposed theoretical model (derived from the MEM) in terms of the causal relationships between its constructs, with the exception of Actual Usage. In particular, this validation tests the predictive and explanatory power of the adapted evaluation method in predicting the likelihood of acceptance of MoS@RT.

To do this, we have chosen regression analysis to evaluate the adapted evaluation model since the hypotheses to be tested are causal relationships between continuous variables. The following levels of significance were used [32]: Not significant: $p > 0.1$; Low: $p < 0.1$; Medium: $p < 0.05$; High: $p < 0.01$; Very high: $p < 0.001$.

1) EFFICIENCY VS. PERCEIVED EASE OF USE

Hypothesis H4 was tested to verify whether the perceptions of Perceived Ease of Use (PEOU) are actually determined by Efficiency when applying the method. A simple regression model was built to perform this analysis in which efficiency was used as the independent (predictor) variable and PEOU as the dependent (predicted) variable. The regression equation resulting from the analysis is as follows:

$$PEOU = 4.044 + (-0.17) * Efficiency \quad (4)$$

The regression model was found not to be significant, with $p > 0.1$ (see Table 6). The R^2 statistic shows that the Efficiency variable is able to explain only 3.2% of the variance in PEOU, indicating that the participants' actual efficiency does not influence their perceptions of ease of use. These results do not allow rejecting $H4_0$ and accepting its alternative hypothesis, meaning that it has been empirically corroborated that PEOU is not determined by Efficiency.

TABLE 6. Simple regression between actual efficiency and perceived ease of use.

Reg. Element	Coef (b)	Std. E.	Std. Coef	t	Sig (p)	R	R^2
Constant	4.044	0.344		11.76	<0.001		
Efficiency	-.017	0.012	-.178	-1.36	0.181	0.178	0.032

2) EFFECTIVENESS VS. PERCEIVED USEFULNESS

H5 was tested to verify whether Perceived Usefulness (PU) perceptions are actually determined by the participants' Effectiveness. Similarly, a simple regression model was built in which effectiveness was used as the independent variable, whereas PU was used as the dependent variable. The equation obtained from the model is as follows:

$$PU = 2.808 + 1.123 * Effectiveness \quad (5)$$

The regression model has a medium significance, with $p < 0.05$ (see Table 7). The R^2 statistic shows that Effectiveness is able to explain 10.6% of the variance of PU, indicating that certain perceptions regarding PU are determined by the participants' effectiveness when applying the method. As expected, the regression coefficient for effectiveness was positive, meaning that the higher the effectiveness values, the higher the PU values. These results allow us to reject $H5_0$ and accept its alternative hypothesis, meaning that we have empirically corroborated that PU is determined by effectiveness.

TABLE 7. Simple regression between actual effectiveness and perceived usefulness.

Reg. Element	Coef (b)	Std. E.	Std. Coef	t	Sig (p)	R	R2
Constant	2.808	0.417		6.738	<0.001		
Effectiveness	1.123	0.451	0.326	0.490	0.016	0.326	0.106

3) PEOU VS. PERCEIVED USEFULNESS

H6 was tested to verify whether Perceived Usefulness (PU) perceptions are actually determined by Perceived Ease of Use (PEOU). Similarly, we built a simple regression model in which the PEOU variable was used as the independent variable, whereas PU was used as the dependent variable. The equation obtained from the regression model is as follows:

$$PU = 2.739 + 0.294 * PEOU \tag{6}$$

The regression model was found to be highly significant, with $p < 0.01$ (see Table 8). The R^2 statistic shows that the Perceived Ease of Use variable is able to explain 16.9% of the variance in PU, indicating that certain perceptions regarding PU are determined by the PEOU. These results allow rejecting $H6_0$ and accepting its alternative hypothesis, meaning that we have empirically corroborated that PU is determined by PEOU.

TABLE 8. Simple regression between perceived ease of use and perceived usefulness.

Reg. Element	Coef (b)	Std. E.	Std. Coef	t	Sig (p)	R	R^2
Constant	2.739	0.343		7.993	<0.001		
PEOU	0.294	0.091	0.411	3.247	0.002	0.411	0.169

4) INTENTION TO USE VS. PERCEIVED USEFULNESS

H7 was tested to verify whether Intention to Use (ITU) perceptions are actually determined by Perceived Usefulness (PU) when applying the method. To perform this analysis, we built a simple regression model in which the PU variable was used as the independent variable, whereas ITU was used as the dependent variable. The equation obtained from the model is as follows:

$$ITU = 0.553 + 0.849 * PU \tag{7}$$

According to Table 9, the regression model was found to be highly significant, with $p < 0.01$. The R^2 statistic shows that the PU variable is able to explain 44.6% of the variance in ITU, which presents a high value given that there could be other factors that influence the intention of the participants of using a method. These results allow rejecting $H7_0$ and accepting its alternative hypothesis, meaning that we have corroborated that ITU is determined by PU.

5) INTENTION TO USE VS. PERCEIVED EASE OF USE

H8 was tested to verify whether Intention to Use (ITU) is actually determined by Perceived Ease of Use (PEOU).

TABLE 9. Simple regression between perceived usefulness and intention to use.

Reg. Element	Coef (b)	Std. E.	Std. Coef	t	Sig (p)	R	R^2
Constant	0.553	0.507		1.090	<0.001		
PU	0.849	0.131	0.7	6.473	0.000	0.668	0.446

Similarly, we built a simple regression model in which the PEOU variable was used as the independent variable, whereas ITU was used as the dependent variable. The equation obtained from the model is as follows:

$$ITU = 2.304 + 0.405 * PEOU \tag{8}$$

According to Table 10, the regression model was found to be highly significant, with $p < 0.01$. The R^2 statistic shows that the PEOU variable can explain 19.7% of the variance in ITU, indicating that the participants' intentions to use the method in the future were determined by their perceptions of the method's ease of use. These results allow rejecting $H8_0$ and accepting its alternative hypothesis, meaning that we have empirically corroborated that ITU is determined by PEOU.

TABLE 10. Simple regression between perceived ease of use and intention to use.

Reg. Element	Coef (b)	Std. E.	Std. Coef	t	Sig (p)	R	R^2
Constant	2.304	0.428		5.381	<0.001		
PEOU	0.405	0.113	0.444	3.574	0.001	0.444	0.197

D. DISCUSSION

The following global conclusions were obtained for each research question:

RQ1: *"Is MoS@RT perceived to be both easy to use and useful to configure the monitoring of cloud services? If so, are the users' perceptions a result of their performance when configuring the quality requirements to be monitored?"* The majority of the participants found MoS@RT quite useful and easy to use when performing the configuration tasks, which are directly related to user interaction. This is supported by their effectiveness when performing the configuration-related tasks, which was very high (91.26%). Therefore, it was found support for hypotheses H1 and H2 related to the participants' perceptions about the method's ease of use and usefulness regarding the configuration of the service monitoring. This result encourages us to continue improving MoS@RT to be applied in industrial contexts for supporting the monitoring configuration of high-level QoS attributes for SaaS, complementing the existing commercial monitoring tools provided by cloud platforms which are more focused on supporting the monitoring of low-level QoS attributes.

Concerning the influence of the users' performance on their perceptions, it has been found that the perceptions on ease of use were not determined by the participants' efficiency (H4 was not confirmed). A possible reason for this could be that the participants perceived some usability

problems with the monitoring configurator. Several participants mentioned this fact in their answers to the open questions of the questionnaire. However, at the time the experiment was performed, the configurator was an early prototype that has been substantially improved since then. In general, the participants provided helpful suggestions, which are being taken into account for improving the tool. Nevertheless, this causal relationship should be analyzed in further replications with other participants. It has also been planned to investigate the influence of other variables that could influence the participants' perceived ease of use.

On the other hand, the results indicated that the perceptions of usefulness were greatly determined by the participant's effectiveness (H5). A possible reason for this could be the fact that the monitoring method and configurator guided the participants in properly specifying how a clause of the SLA (expressed as NFRs) can be mapped into specific quality attributes, metrics and how these metrics could be measured at runtime by using performance counters provided by the selected cloud platform. The participants indicated in the questionnaire that they found the SaaS Quality Model quite useful for monitoring application-specific quality requirements as it guides the definition of metrics and indicators by combining different low-level metrics (e.g., downtime).

RQ2: "*Is there an intention to use MoS@RT* to configure the monitoring of cloud services *in the future? If so, is the intention to use it a result of the perceptions experienced by subjects when configuring the quality requirements to be monitored?*" The majority of the respondents were very positive about the use of MoS@RT for configuring cloud services monitoring in the future, with a mean = 3.8, which was also supported by the open questions enclosed in the questionnaire. Hypothesis H3 was confirmed, signifying that the participants have the intention to use MoS@RT in the future to support configuration tasks. It has been shown that there may be other factors that could affect people's decision when using a cloud monitoring method or tool (e.g., integrated platform tools, organization standards, licenses). However, these are uncontrollable factors. The objective here was to select variables that can be tested and controlled, such as the behavior of participants using a cloud monitoring method. It has been considered perceived ease of use and perceived usefulness, because they are the most important factors as regards explaining system acceptance/usage [15], [36]. The objective of the adapted evaluation method was to provide a basis that can be used to trace the impact of external variables on internal beliefs, attitudes and intentions. The results provide further evidence of the perceived efficacy of MoS@RT to support the configuration of cloud services and the easy change of NFRs and/or metrics due to renegotiations of SLAs.

With regard to the influence of the users' perceptions on their intentions, there was found support to H6, H7 and H8, meaning that the participants' perception of ease of use and usefulness determined their intentions to use MoS@RT in the future to configure the monitoring of cloud services.

The global results of the regression analysis performed to validate the adapted evaluation method are summarized in Figure 11. All the causal relationships between the model construts were confirmed with good levels of confidence, with the exception of the relationship between Efficiency and PEOU that could not be confirmed with our experimental data. We need to perform replications of this experiment in other contexts to further verify if PEOU can determined by Efficiency.

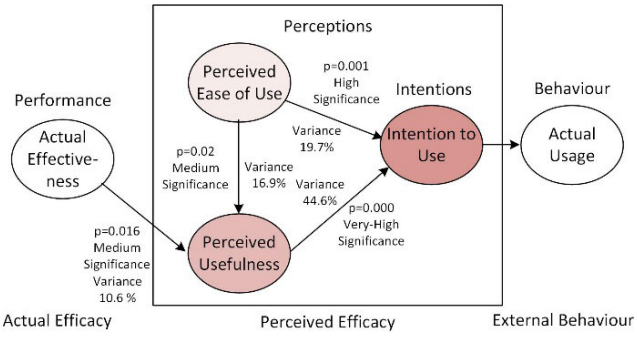

FIGURE 11. Results of the application of MEM to MoS@RT.

In general, our findings are consistent with the results obtained by Moody [36]. However, while the regression results were found to be significant, the variance value is not too high for high data dispersion in the causal models. The portion of the variance explained by the model was lower than 34%, and the extreme values cannot be predicted with high accuracy. Nevertheless, these results constitute the first empirical test of the evaluation model proposed in Section 5. As future work, we plan to investigate the influence of other performance-based and perception-based variables on predicting the acceptance of MoS@RT in practice for supporting the configuration of cloud services by replicating the experiment in other contexts, with other services and more experienced participants. Finally, as a next step, it is planned to evaluate the monitoring data and their correctness and effectiveness, taking into account methods to calculate the sampling frequency to avoid possible system overload. The causal-connection characteristic of the models@runtime will be used in order to self-adapt the sampling frequency depending on the current state of services.

E. THREATS TO VALIDITY

This section discusses the possible threats that can affect the validity of the results obtained.

1) CONCLUSION VALIDITY

To control the risk of variation owing to individual differences being larger than those owing to the treatment, a homogeneous group of subjects was selected. Another risk in the experiment implementation was that the participants were trained on one day, and the experiment was run the following week, signifying that they might have forgotten some details about the execution of the method. This threat was reduced by performing a short training before the experiment.

Another issue is the time spent on the training session. It is established that in a practical setting, more time may be required. However, owing to time constraints (the experiment was conducted as part of a course), it was decided to use a guide that summarizes the method's application with examples. The participants used this guide in the experiment. Nevertheless, it has been planned to replicate this experiment to verify whether this issue might impact the interpretation of the results.

2) CONSTRUCT VALIDITY

The main threat to construct validity is that of reflecting the efficacy in how the NFRs have been configured to be monitored and whether the monitoring results reflected the real state of the services running on the Cloud, providing the expected cloud service assessment. The subjective variables are based on the Method Evaluation Model (MEM) [36], a well-known and empirically validated model for evaluating information technologies. The main threat is, therefore, the reliability of the questionnaire. An analysis of Cronbach's alpha reliability test was carried out for each set of questions related to each subjective variable to evaluate the questionnaire's reliability. These were greater than the minimum acceptance threshold $\alpha = 0.70$, where the Crombach's α of PEOU is 0.834, PU is 0.776, and ITU is 0.820.

3) INTERNAL VALIDITY

The threats to internal validity are relevant in those studies that attempt to establish causal relationships. The main threats to the internal validity were: the participants' experience, author bias, and the understandability of the cloud monitoring method and supporting tool.

To reduce the threat related to the participant's experience, a representative training example was prepared, which showed every step of the process and provided the users with an in-depth understanding of cloud services monitoring and how to monitor services using this approach. Both author bias and the bias produced by the material's understandability were reduced during a pilot experiment's execution. A group or researchers, experts in the field evaluated the experimental material to reduce possible errors or misunderstandings related to the experiment. Monitoring tool bias was reduced by its validation in the pilot experiment and successive tests to improve the tool's usability.

4) EXTERNAL VALIDITY

External validity refers to the approximate truth of conclusions involving generalizations within different contexts. The main threat to external validity is the representativeness of the results that might be affected by the evaluation's design, the participant context selected, and the size and complexity of the tasks. The evaluation design might have had an impact on the generalization of the results owing to the complexity of the cloud platform, its particular characteristics, the tools used to retrieve raw data, and the NFRs to be monitored. It was attempted to reduce this issue by selecting a popular and commonly used platform, which shares concepts with other platforms, and considers a very common scenario from which to gather raw data. Furthermore, the selected NFRs, were representative requirements of cloud services. Concerning the participants' experience, the quasi-experiment was conducted with Computer Science undergraduates, who attended a Software Quality course and had a good knowledge of quality models and metrics.

Moreover, they were trained in the use of the monitoring approach and tool for a reasonable amount of time. It will, however, be necessary to perform further experiments with professional industrial participants. The size and complexity of the tasks might also have affected the external validity. It was attempted to propose a set of experimental tasks with a sufficient level of complexity, given the sessions' time constraints.

The experiment performed covered only the activity related to the monitoring configuration. Further experiments must be perfomed in order to evaluate our monitoring method as a whole. In such experiments, the participants should not only perform the configuration of the services monitoring to generate the runtime quality model, but they should also use and evaluate the Monitoring & Analysis Middleware to check i) if it produces accurate and efficient measures for the stated NFRs; and ii) allow the visualization of monitoring results in real time in an flexible and customizable way.

Finally, we are aware that the proposed evaluation method cannot be generalized for its use with any cloud service monitoring method. It may be useful to evaluate other approaches that pursue the same objectives of MoS@RT, but this should be tested empirically in further experiments.

VIII. CONCLUSION AND FUTURE WORK

In this paper, a monitoring infrastructure supporting the MoS@RT method instantiated to Microsoft Azure was introduced. The method provides as a flexible solution for monitoring SLAs for cloud services using models at runtime, which allows the monitoring requirements to be changed at runtime without the modification of the underlying infrastructure. Moreover, we presented a quasi-experiment to evaluate the perceived efficacy of the infrastructure for supporting the monitoring configuration of SaaS applications. The method used to validate this approach was tailored using the Method Evaluation Model as a basis, which uses both aspects of method success: actual performance and likelihood of acceptance in practice. It has been defined performance-based variables (i.e., efficiency and effectiveness) as influencing factors for the perception-based variables (i.e., Perceived Ease of Use, Perceived Usefulness, and Intention to Use).

The main results obtained from the analysis of the data gathered revealed that: (i) the majority of the participants found that the MoS@RT method is quite useful and easy to use to configure the monitoring of cloud services; (ii) the majority of the participants were also very optimistic about the use of MoS@RT method in the future; (iii) the performance of the participants in the quasi-experiment determined

their positive perceptions; (iv) the perceptions determine the intention to use MoS@RT.

In summary, the proposed method based on models at runtime was found to be a suitable and flexible approach to configure the monitoring of SaaS applications and to easily change the monitoring requirements, in the context of the experiment. It is necessary to be aware that this study provides only preliminary results. There is a need for more empirical studies with which to test the monitoring method as a whole, as well as in other settings. Nevertheless, this study has value as a pilot study to test our approach concerning user interactions and expectations related to the configuration of cloud services monitoring. Most importantly, we believe that our study provides new insights into the problem of decoupling the monitoring configuration from the actual monitoring of the service. It is mainly achieved through the use of models at runtime, which allows: (i) the level of abstraction for raw data obtained from the service execution to be raised, and (ii) the monitoring directives needed to manage the services data obtained from the underlying cloud platform to be dynamically changed.

We are currently improving our monitoring infrastructure to integrate other third-party solutions using APIs and plugins (e.g., RackSpace, Amazon CloudWatch) as part of the monitoring configurator and middleware to allow multi-cloud monitoring. This will permit other commercial or academic initiatives to deal with more complex cloud applications manteining the flexibility of our approach to monitor high-level quality requirements. We also plan to deploy our middleware on other cloud platforms such as Amazon Web Services and Google Cloud Platform and compare the entire monitoring method with other commercial solutions to contrast its flexibility, portability, and efficiency. We are also exploring self-adaptation mechanisms that can be used in the monitoring infrastructure. The calculation formulas in the model at runtime may have the knowledge to choose the best possible alternative formula according to the data or utility services available at a particular moment.

Since more and more organizations are gradually moving their workloads to public or hybrid clouds, cloud adoption continues to grow. Changes in requirements, business priorities, underlying technologies, or consumer demands may require changes in the SLAs and customers' expectations. In this context, there is a growing need for more flexible and possible self-adapting monitoring methods and tools.

REFERENCES

[1] P. Mell and T. Grance, "The NIST definition of cloud computing," Nat. Inst. Standards Technol., Gaithersburg, MD, USA, Tech. Rep. SP 800-145, 2011, p. 7.

[2] S. A. Baset, "Cloud SLAs," *ACM SIGOPS Oper. Syst. Rev.*, vol. 46, no. 2, pp. 57–66, Jul. 2012, doi: 10.1145/2331576.2331586.

[3] Y. Jiang, C.-S. Perng, T. Li, and R. Chang, "Self-adaptive cloud capacity planning," in *Proc. IEEE 9th Int. Conf. Services Comput. (SCC)*, Jun. 2012, pp. 73–80, doi: 10.1109/SCC.2012.8.

[4] K. Fatema, V. C. Emeakaroha, P. D. Healy, J. P. Morrison, and T. Lynn, "A survey of cloud monitoring tools: Taxonomy, capabilities and objectives," *J. Parallel Distrib. Comput.*, vol. 74, no. 10, pp. 2918–2933, Oct. 2014, doi: 10.1016/j.jpdc.2014.06.007.

[5] D. Dawson, R. Desmarais, H. M. Kienle, and H. A. Müller, "Monitoring in adaptive systems using reflection," in *Proc. Int. Workshop Softw. Eng. Adapt. Self-Managing Syst. (SEAMS)*, 2008, pp. 81–88, doi: 10.1145/1370018.1370033.

[6] C. Muller, M. Oriol, X. Franch, J. Marco, M. Resinas, A. Ruiz-Cortes, and M. Rodriguez, "Comprehensive explanation of SLA violations at runtime," *IEEE Trans. Services Comput.*, vol. 7, no. 2, pp. 168–183, Apr. 2014, doi: 10.1109/TSC.2013.45.

[7] A. Taherkordi, F. Zahid, Y. Verginadis, and G. Horn, "Future cloud systems design: Challenges and research directions," *IEEE Access*, vol. 6, pp. 74120–74150, 2018, doi: 10.1109/ACCESS.2018.2883149.

[8] N. Shahida, M. Jamail, R. Atan, R. Abdullah, and M. Y. Said, "Development of SLA monitoring tools based on proposed DMI in cloud computing," *Trans. Mach. Learn. Artif. Intell.*, vol. 3, no. 1, pp. 1–7, 2015. [Online]. Available: https://journals.scholarpublishing.org/index.php/TMLAI/issue/view/45/84, doi: 10.14738/tmlai.31.841.

[9] D. A. Tamburri, M. Miglierina, and E. D. Nitto, "Cloud applications monitoring: An industrial study," *Inf. Softw. Technol.*, vol. 127, Nov. 2020, Art. no. 106376, doi: 10.1016/j.infsof.2020.106376.

[10] P. Cedillo, J. Gonzalez-Huerta, E. Insfrán, and S. Abrahao, "Towards monitoring cloud services using models@run time," in *Proc. Workshop Models@run.time (MODELS)*, 2014, pp. 31–40.

[11] P. Cedillo, J. Gonzalez-Huerta, S. Abrahao, and E. Insfrán, "A monitoring infrastructure for the quality assessment of cloud services," in *Transforming Healthcare Through Information Systems* (Lecture Notes in Information Systems and Organisation), vol. 17, D. Vogel, X. Guo, H. Linger, C. Barry, M. Lang, and C. Schneider, Eds. Springer, 2016, pp. 17–32, doi: 10.1007/978-3-319-30133-4_2.

[12] H. Giese, N. Bencomo, L. Pasquale, A. J. Ramirez, P. Inverardi, S. Wätzoldt, and S. Clarke, "Living with uncertainty in the age of runtime models," in *Models@runtime: Foundations, Applications and Roadmaps* (Lecture Notes in Computer Science), vol. 8378, N. Bencomo, R. France, B. H. C. Cheng, and U. Abmann, Eds. Cham, Switzerland: Springer, 2014, pp. 47–100, doi: 10.1007/978-3-319-08915-7_3.

[13] P. Cedillo, J. Jimenez-Gomez, S. Abrahao, and E. Insfran, "Towards a monitoring middleware for cloud services," in *Proc. IEEE Int. Conf. Services Comput.*, New York, NY, USA, Jun. 2015, pp. 451–458, doi: 10.1109/SCC.2015.68.

[14] D. L. Moody, "The method evaluation model: A theoretical model for validating information systems design methods," in *Proc. 11th Eur. Conf. Inf. Syst. (EICS)*, Naples, Italy, vol. 79, 2003, pp. 1–17. [Online]. Available: https://aisel.aisnet.org/ecis2003/79

[15] F. D. Davis, "Technology acceptance model for empirically testing new end-user information systems: Theory and results," Ph.D. dissertation, Sloan School Manage., Massachusetts Inst. Technol., Cambridge, MA, USA, 1986.

[16] V. C. Emeakaroha, T. C. Ferreto, M. A. S. Netto, I. Brandic, and C. A. F. De Rose, "CASViD: Application level monitoring for SLA violation detection in clouds," in *Proc. IEEE 36th Annu. Comput. Softw. Appl. Conf.*, Jul. 2012, pp. 499–508, doi: 10.1109/COMPSAC.2012.68.

[17] G. Aceto, A. Botta, W. de Donato, and A. Pescapè, "Cloud monitoring: A survey," *Comput. Netw.*, vol. 57, no. 9, pp. 2093–2115, Jun. 2013, doi: 10.1016/j.comnet.2013.04.001.

[18] H. J. Syed, A. Gani, R. W. Ahmad, M. K. Khan, and A. I. A. Ahmed, "Cloud monitoring: A review, taxonomy, and open research issues," *J. Netw. Comput. Appl.*, vol. 98, pp. 11–26, Nov. 2017, doi: 10.1016/j.jnca.2017.08.021.

[19] K. Alhamazani, R. Ranjan, K. Mitra, F. Rabhi, P. P. Jayaraman, S. U. Khan, A. Guabtni, and V. Bhatnagar, "An overview of the commercial cloud monitoring tools: Research dimensions, design issues, and state-of-the-art," *Computing*, vol. 97, no. 4, pp. 357–377, Apr. 2015, doi: 10.1007/s00607-014-0398-5.

[20] A. Keller and H. Ludwig, "The WSLA framework: Specifying and monitoring service level agreements for Web services," *J. Netw. Syst. Manage.*, vol. 11, no. 1, pp. 57–81, 2003, doi: 10.1023/A:1022445108617.

[21] J. Shao, H. Wei, Q. Wang, and H. Mei, "A runtime model based monitoring approach for cloud," in *Proc. IEEE 3rd Int. Conf. Cloud Comput.*, Jul. 2010, pp. 313–320, doi: 10.1109/CLOUD.2010.31.

[22] K. J. Modi, D. P. Chowdhury, and S. Garg, "Automatic cloud service monitoring and management with prediction-based service provisioning," *Int. J. Cloud Comput.*, vol. 7, no. 1, pp. 65–82, 2018, doi: 10.1504/IJCC.2018.091684.

[23] A. Shatnawi, M. Orrù, M. Mobilio, O. Riganelli, and L. Mariani, "Cloudhealth: A model-driven approach to watch the health of cloud services," in *Proc. 1st Int. Workshop Softw. Health*, May 2018, pp. 40–47, doi: 10.1145/3194124.3194130.

[24] K. Alhamazani, R. Ranjan, P. Prakash Jayaraman, K. Mitra, C. Liu, F. Rabhi, D. Georgakopoulos, and L. Wang, "Cross-layer multi-cloud real-time application QoS monitoring and benchmarking as-a-service framework," *IEEE Trans. Cloud Comput.*, vol. 7, no. 1, pp. 48–61, Jan. 2019, doi: 10.1109/TCC.2015.2441715.

[25] X. Lu, J. Yin, N. N. Xiong, S. Deng, G. He, and H. Yu, "JTangCMS: An efficient monitoring system for cloud platforms," *Inf. Sci.*, vols. 370–371, pp. 402–423, Nov. 2016, doi: 10.1016/j.ins.2016.06.009.

[26] L. Bodenstaff, A. Wombacher, and M. Reichert, "Empirical validation of MoDe4SLA; Approach for managing service compositions," in *Proc. 14th Int. Conf. Bus. Inf. Syst.*, no. 612, 2011, pp. 98–110, doi: 10.1007/978-3-642-21863-7_9.

[27] S. Meng and L. Liu, "Enhanced monitoring-as-a-service for effective cloud management," *IEEE Trans. Comput.*, vol. 62, no. 9, pp. 1705–1720, Sep. 2013, doi: 10.1109/TC.2012.165.

[28] J. Montes, A. Sánchez, B. Memishi, M. S. Pérez, and G. Antoniu, "GMonE: A complete approach to cloud monitoring," *Future Gener. Comput. Syst.*, vol. 29, no. 8, pp. 2026–2040, Oct. 2013, doi: 10.1016/j.future.2013.02.011.

[29] J. Povedano-Molina, J. M. Lopez-Vega, J. M. Lopez-Soler, A. Corradi, and L. Foschini, "DARGOS: A highly adaptable and scalable monitoring architecture for multi-tenant clouds," *Future Gener. Comput. Syst.*, vol. 29, no. 8, pp. 2041–2056, Oct. 2013, doi: 10.1016/j.future.2013.04.022.

[30] X. Guerron, S. Abrahão, E. Insfran, M. Fernández-Diego, and F. González-Ladrón-De-Guevara, "A taxonomy of quality metrics for cloud services," *IEEE Access*, vol. 8, pp. 131461–131498, 2020, doi: 10.1109/ACCESS.2020.3009079.

[31] Amazon Elastic Compute. (2014). *Sending Performance Counters to CloudWatch and Logs to CloudWatch Logs*. [Online]. Available: http://docs.aws.amazon.com/AWSEC2/latest/WindowsGuide/ec2-configuration-cwl.html

[32] Microsoft Azure. (2014). *What is Azure?* [Online]. Available: http://msdn.microsoft.com/en-us/library/azure/dd163896.aspx

[33] Microsoft MSDN. (2016). *Introducing Visual Studio.NET*. Accessed: May 10, 2019. [Online]. Available: https://msdn.microsoft.com/en-us//library/aa291755(v=vs.71).aspx

[34] MSDN Library. (2014). *Collect Logging Data by Using Azure Diagnostics*. Accessed: Feb. 8, 2015. [Online]. Available: https://msdn.microsoft.com/en-us/library/azure/gg433048.aspx

[35] (2011). *NCalc—Mathematical Expressions Evaluator for.NET*. Accessed: May 12, 2016. [Online]. Available: https://ncalc.codeplex.com/

[36] D. L. Moody, "Dealing with complexity: A practical method for representing large entity relationship models," Ph.D. dissertation, Dept. Inf. Syst., Univ. Melbourne Melbourne, VIC, Australia, 2001.

[37] S. Abrahão, E. Insfran, J. A. Carsí, and M. Genero, "Evaluating requirements modeling methods based on user perceptions: A family of experiments," *Inf. Sci.*, vol. 181, no. 16, pp. 3356–3378, Aug. 2011, doi: 10.1016/j.ins.2011.04.005.

[38] N. Rescher, *The Primacy of Practice*. Cambridge, U.K.: Cambridge Univ. Press, 1973.

[39] A. Bertolino, A. Calabrò, F. Lonetti, A. Di Marco, and A. Sabetta, "Towards a model-driven infrastructure for runtime monitoring," in *Proc. 3rd Int. Workshop Softw. Eng. Resilient Syst.*, vol. 6968, 2011, pp. 130–144, doi: 10.1007/978-3-642-24124-6_13.

[40] H. Foster and G. Spanoudakis, "Dynamic creation of monitoring infrastructures," in *Service Level Agreements for Cloud Computing*, P. Wieder, J. M. Butler, W. Theilmann, and R. Yahyapour, Eds. New York, NY, USA: Springer, 2011, pp. 123–138.

[41] P. Wieder, J. M. Butler, W. Theilmann, and R. Yahyapour, *Service Level Agreements for Cloud Computing*. Berlin, Germany: Springer, 2011. [Online]. Available: https://link.springer.com/book/10.1007/978-1-4614-1614-2, doi: 10.1007/978-1-4614-1614-2.

[42] Amazon Cloud Services. (2016). *Amazon Cloud Watch*. Accessed: Jul. 2, 2016. [Online]. Available: https://aws.amazon.com/es/cloudwatch/

[43] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in Software Engineering*. New York, NY, USA: Springer, 2012.

[44] S. Herold, H. Klus, Y. Welsch, C. Deiters, A. Rausch, R. Reussner, K. Krogmann, H. Koziolek, R. Mirandola, B. Hummel, M. Meisinger, and C. Pfaller, "CoCoMe—The common component modeling example," in *The Common Component Modeling Example* (Lecture Notes in Computer Science), vol. 5153. Berlin, Germany: Springer-Verlag, 2008, pp. 16–53.

PRISCILA CEDILLO received two master's degrees, the former master's degree in telematics from the Universidad de Cuenca, and the second master's degree in software engineering, information systems, and formal methods from the Universitat Politécnica de Valéncia (UPV), and the Ph.D. degree in computer science from UPV, in 2017. She received a scholarship from the Senescyt for her Ph.D. studies from the Fundación Carolina for a Postdoctoral Research stay. She has been an Associate Professor with the Universidad de Cuenca, Ecuador, since 2009. Her main research interests include model-driven engineering, cloud computing, software quality, and the Internet of Things (IoT).

EMILIO INSFRAN received the M.Sc. degree in computer science from Cantabria University, Spain, in 1994, and the Ph.D. degree from the Universitat Politécnica de Valéncia (UPV), Spain, in 2003. He worked as a Visiting Researcher with the Université Catholique de Louvain, Belgium, in 2017, and Software Engineering Institute (SEI), Carnegie Mellon University, Pittsburgh, PA, USA, in 2012. He also performed research stays at the University of Twente, Netherlands, Brigham Young University, Utah, Provo, UT, USA, and the University of Porto Alegre, Brazil. He is currently working as an Associate Professor with the Department of Information Systems and Computation (DISC), UPV. He has coauthored more than 150 journal and conference papers. He has worked on a number of national and international research projects and several technology transfer projects with companies in Spain. His research interests include cloud service architectures, DevOps, model-driven development, requirements engineering, and software quality.

SILVIA ABRAHÃO received the Ph.D. degree in computer science from the Universitat Politécnica de Valéncia (UPV), Spain, in 2004. She was a Visiting Professor with Carnegie Mellon Software Engineering Institute, from 2010 to 2012, the Université Catholique de Louvain, from 2007 to 2017, and Ghent University, in 2004. She is currently an Associate Professor with UPV. She has (co)authored over 150 peer-reviewed publications. Her main research interests include quality assurance in model-driven engineering, empirical assessment of software modeling approaches, software quality, usability integration into software development, and cloud services monitoring and adaptation. She is a member of the Editorial Board of the *Software and System Modeling* (SoSyM) journal. She currently leads the Spanish Network of Excellence on Software Quality and Sustainability. She is an Associate Editor of IEEE Software, where she is responsible for software quality.

JEAN VANDERDONCKT received the M.Sc. degree in mathematics, the M.Sc. degree in computer science, and the Ph.D. degree from the University of Namur, Belgium, in 1987, 1989, and 1997, respectively. He was a Visiting Associate Professor with Stanford University, in 2000. He is a permanent Invited Professor with the Universitat Politécnica de Valencia. He is currently a Full Professor with the Louvain School of Management, Université Catholique de Louvain, Belgium, where he has been leading the Louvain Interaction Laboratory, since 1998. His research interests include human–computer interaction (HCI), engineering interactive computing systems (EICS), intelligent user interfaces (IUI), usability engineering, and software engineering. He is an Associate Editor of *ACM Transactions on Interactive Intelligent Systems* (TiiS) and the Co-Editor-in-Chief of the Springer Series of *Human–Computer Interaction* and the Springer Briefs in *Human–Computer Interaction*.