Towards a Monitoring Middleware for Cloud Services

Priscila Cedillo, Javier Jimenez-Gomez, Silvia Abrahao, Emilio Insfran
Department of Computer Systems and Computation
Universitat Politècnica de València
Camino de Vera, s/n, 46022, Valencia, Spain
icedillo@dsic.upv.es, jajimgme@inf.upv.es, {sabrahao, einsfran}@dsic.upv.es

***Abstract*— Cloud Computing represents a new trend in the development and use of software. Many organizations are currently adopting the use of services that are hosted in the cloud by employing the Software as a Service (SaaS) model. Services are typically accompanied by a Service Level Agreement (SLA), which defines the quality terms that a provider offers to its customers. Many monitoring tools have been proposed to report compliance with the SLA. However, they have some limitations when changes to monitoring requirements must be made and because of the complexity involved in capturing low-level raw data from services at runtime. In this paper, we propose the design of a platform-independent monitoring middleware for cloud services, which supports the monitoring of SLA compliance and provides a report containing SLA violations that may help stakeholders to make decisions regarding how to improve the quality of cloud services. Moreover, our middleware definition is based on the use of models@run.time, which allows the dynamic change of quality requirements and/or the dynamic selection of different metric operationalizations (i.e., calculation formulas) with which to measure the quality of services. In order to demonstrate the feasibility of our approach, we show the instantiation of the proposed middleware that can be used to monitor services when deployed on the Microsoft Azure© platform.**

Cloud Computing; Software as a Service; Monitoring; Middleware; Quality of Service; Models@run.time

I. INTRODUCTION

Cloud Computing represents a new trend in the development and use of software systems. Cloud platforms allow the provision of infrastructure as a service (IaaS), platform as a service (PaaS), or software as a service (SaaS). The business model that cloud computing offers has a huge number of advantages (e.g., scalability, elasticity, availability, pay-as-you-go flexible charging). These advantages could be translated into quality requirements that should be fulfilled during the provision of the service. The provision of SaaS includes the delivery of software applications or services to clients via the Internet. A SaaS model may have a large number of tenants and each tenant may have hundreds of thousands of users, meaning that a SaaS infrastructure can support millions of users with scalable performance [1]. The terms under which a service is provided must be expressed by using Service Level Agreements (SLAs).

Each service is typically accompanied by an SLA that defines the minimal guarantees that a cloud provider offers to its customers [2] (e.g., ensure web application server latency to be less than 100 ms). Cloud providers usually offer some performance guarantees for the provision of services and leave the detection of SLA violations to the customer [2]. Penalties are established when the service quality violates the SLA. Both the underestimation of provisioning and overestimation of de-provisioning lead to penalties [3]. Moreover, many cloud providers do not automatically credit the customer for SLA violations and leave the burden of providing evidence of any such violation to the customer [2]. The provision of an SLA violation report that helps both clients and providers to understand the real behavior of services and SLA compliance would be useful. Unfortunately, existing solutions have several drawbacks, as is reported by Muller et al. [4]: the SLAs that they support are not sufficiently expressive to model real-world scenarios; the monitoring configuration is coupled with a given SLA specification; and explanations of the violations are difficult to understand and potentially inaccurate.

On the other hand, the deployment and execution of software systems in highly dynamic infrastructures (i.e., clouds) lead to a new set of challenges and requirements with regard to monitoring. A monitoring system should consequently have the flexibility to be adapted or changed according to new monitoring requirements and should keep up when a service scales up or down dynamically [5].

We believe that Model-Driven Engineering (MDE) may be a solution that provides the flexibility required to monitor cloud services. However, establishing the whole set of Non-Functional Requirements (NFRs) to be monitored at design time is not always possible owing to renegotiations of SLAs or the addition of new NFRs to be monitored. It is therefore not sufficient to set requirements statically because they may change at run-time [6]. Baresi and Ghezzi [7] advocate that future Software Engineering research should be focused on providing intelligent support for software at run-time, thus crossing the current rigid boundary between development-time and run-time. It is therefore necessary to define approaches that will permit the run-time monitoring of cloud services, the addition of new quality requirements to be monitored, and the selection of the most appropriate metric operationalization [1] depending on the cloud capabilities,

[1] The operationalization of a metric consists of establishing a mapping between the generic specification of the metric and the concepts that are represented in the software artifacts to be measured [20].

without interrupting the execution of services. This challenge can be tackled by using models@run.time [8] [9], which, by means of dynamic reflection mechanisms, allows the calculation formula used for specific metrics to be changed in order to satisfy quality requirements as well as to define new quality requirements at run-time.

In previous studies, we have proposed the definition of a monitoring process for cloud services by using models@run.time [10]. This monitoring process provides an SLA violation report, which contains the non-compliance with the SLA and additional NFRs that may be of interest to the service provider. In this paper, we propose the design of a platform-independent monitoring middleware for cloud services, which supports the monitoring of SLA compliance and provides a report containing SLA violations that may help stakeholders to make decisions regarding how to improve the quality of cloud services. Furthermore, our middleware definition is based on the use of models@runtime, which allows the dynamic change of quality requirements and/or the dynamic selection of different metric operationalizations (i.e., calculation formulas) with which to measure the quality of services. In order to demonstrate the feasibility of our approach, we show the instantiation of the proposed middleware that can be used to monitor services when deployed on the Microsoft Azure© platform.

The paper is organized as follows. In Section II, we discuss existing approaches and platforms that are used to monitor cloud services. In Section III, we describe the monitoring process and introduce the problem that the proposed middleware is addressing. In Section IV, we present the architecture of the platform-independent monitoring middleware. In Section V, we present the instantiation of the proposed middleware for monitoring services that are deployed on the Microsoft Azure © platform. Finally, in Section VI, we present our conclusions and further work.

II. Related Work

Recent research and industry efforts have focused on developing monitoring techniques, platforms, and frameworks that can assist cloud providers in tracking the SLA violations of certain service quality requirements. In this section, we analyze the flexibility and maintainability of these solutions when a monitoring requirement needs to be added or modified.

Currently, many public cloud providers provide their consumers with the ability to monitor their cloud services using available monitoring tools for CPU, storage, and network [11]. These tools are often tightly integrated with their other cloud solutions. For example, CloudWatch (offered by Amazon) is a monitoring tool that enables consumers to manage and monitor their applications residing on AWS EC2 (CPU) services. However, this monitoring tool does not have the ability to monitor a service component that may reside on the infrastructure of other cloud providers such as GoGrid and Azure. Another limitation of tools of this kind is that they are only concerned with monitoring the quality of the service attributes for hardware resources (CPU, storage, and network) and lack the ability to monitor application-specific QoS attributes and SLA requirements (i.e., latency, performance, security). In addition, the majority of these commercial tools (e.g., CloudWatch, LogicMonitor) are not flexible enough to allow a service provider to extend the provided QoS attributes in order to monitor the fulfillment of SLAs.

Some research works have also focused on surveying the current state of monitoring tools [12] or providing solutions for cloud service monitoring [4], [13], [14]. Fatema et al. [12] survey the range of monitoring tools that are currently in use in order to gain an insight into their technical capabilities. They also identify the desired capabilities of monitoring tools used to serve different cloud operational areas from the perspective of both providers and customers; the authors then present a taxonomy of the capabilities identified and analyze the tools available using these capabilities as a basis. They conclude that the monitoring tools were mostly not designed with the cloud in mind and must be extended or redesigned to be useable in cloud environments. Consequently, there is a lack of support for monitoring desirable capabilities that are specific to cloud platforms (e.g., elasticity, scalability).

Keller and Ludwig [13] describe a framework named WSLA that can be used to specify and monitor SLAs for web services. They have developed a prototype of a WSLA compliance monitor. However, they do not study how to deal with different operationalizations in order to provide flexibility to their measurement service.

Emeakaroha et al. [14] 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 tools for resource allocation, scheduling, and deployment. The monitoring infrastructure receives instructions to monitor applications from the SLA management framework and delivers the monitored information. It uses agents and an SMTP protocol to collect data from the cloud services. However, this approach does not have a flexible means to change the requirements and metrics to be monitored at run-time. Shao et al. [15] propose a runtime model for cloud monitoring (RMCM), which denotes a representation of a running cloud by focusing on common monitoring concerns. However, they do not mention specific non-functional characteristics for SaaS cloud environments and their metrics (e.g. scalability, availability, elasticity, etc.). Furthermore, they do not provide an SLA violation report and leave addressing non-functional requirements from SLA as future work. Finally, Müller et al. [4] have designed and implemented SALMonADA, which is a service-based system for monitoring and analyzing SLAs in order to provide an explanation for violations. The SLA is described using a monitoring management document (MMD) to be consumed by the monitoring infrastructure; however, these authors do not help stakeholders in selecting alternative operationalizations depending on the platform, and require advanced users with knowledge about metrics and specific platforms.

Overall, these monitoring solutions have focused on some specific quality attributes (e.g., performance) and some of them lack mechanisms to aggregate multiple quality attributes or parameters for a service consumer, which is a critical aspect of monitoring. To the best of our knowledge, there is a need for approaches that monitor the specific non-functional characteristics of cloud services and that allow the flexibility of adding and modifying monitoring requirements at run-time. These changes in monitoring requirements can be due to the renegotiation of SLAs or result from the need to know the quality characteristics of a service that was not of interest when the monitoring requirements were established.

III. Monitoring Process Overview

This section presents the process that is supported by the monitoring middleware. It contains a set of tasks that can be used to monitor, analyze, and report SLA violations. It may also be useful to service providers for monitoring additional NFRs that are not specified in the SLA but that might be of interest during the service provision (e.g., detecting variations in the service performance).

Figure 1 provides an overview of the process. The first step of the process is the Monitoring Configuration task where the NFRS are specified as well as the procedure to collect data from the cloud service. The inputs for this activity are the NFRs to be monitored, which are represented by using a Monitoring Requirements Model. Since this model mainly contains the SLA terms, we use the WSLA Language [16] to express the SLA terms in a standardized manner. The generation of the Monitoring Requirements Model can be supported by the SaaS Quality Model in order to classify and choose the appropriate operationalizations that are needed to monitor the NFRs contained in the SLA. The NFR classification is built in compliance with the SaaS Quality Model, which is aligned with the ISO/IEC 25010 standard (SQuaRE) [17], and it contains all the characteristics, sub-characteristics, attributes, and metrics that are used to measure the quality of SaaS. The output of the Monitoring Configuration task is the Runtime Quality Model, which is a model@run.time that contains the monitoring directives. Also, the matching between non-functional requirements and raw data to be obtained from the cloud takes place in the Monitoring Configuration task, and is included in the Runtime Quality Model. This latter model is used in the Measurement Process task to calculate all of the metrics that are involved in the monitoring process. The SaaS Quality Meta-model and the Runtime Quality Meta-model have been designed by considering quality specifications and reports, which study the meta-classes that are involved in the quality measurement [18][19].

The second step of the process is the Measurement Process task, which captures raw data by using different retrieval scenarios based on the suitability of the technique for gathering quality data from cloud services (e.g., performance counters, custom counters, wrappers, APIs). In the Measurement Process Task, the metrics are calculated and the process is fed with useful and filtered information, which is used by the Analyze Results task. In the third step, the Analyze Results task uses the information generated by the Measurement Process, compares it with the non-functional requirement thresholds specified in the Monitoring Requirements Model, and creates an SLA Violations Report in which any non-compliances are described. Further information on the monitoring process can be found in [10].

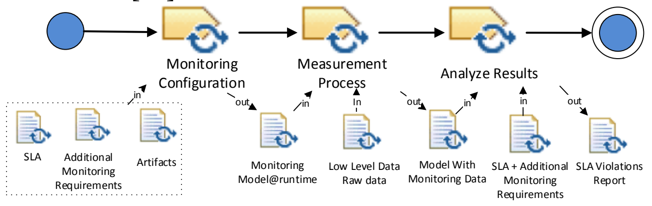


Figure 1. Monitoring Process Overview

IV. Monitoring Middleware Architecture

This section presents the platform-independent middleware architecture. This architecture allows the identification of SLA violations of cloud services by decoupling the specification of the quality attributes that must be evaluated from the monitoring process itself. Figure 2 shows the different elements of the monitoring architecture.

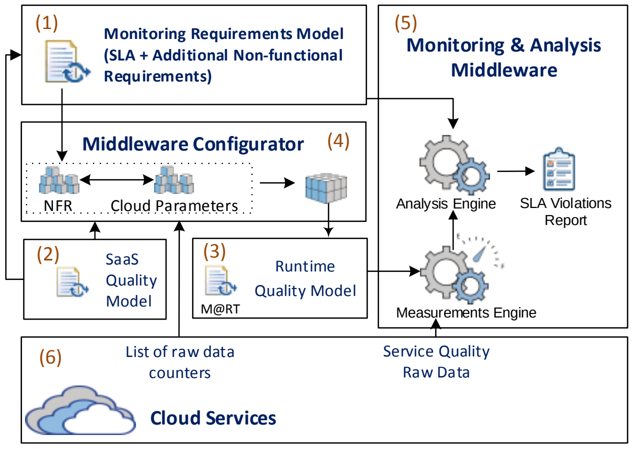


Figure 2. Monitoring Configurator & Middleware

Three models are used to configure the monitoring middleware (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 constraints established in the SLA together with the corresponding thresholds, which should be evaluated. The SaaS Quality Model (see Figure 2 (2)) represents the decomposition of SaaS quality characteristics into measurable quality attributes and the different metric operationalization alternatives that can be used during the service monitoring process. The operationalization of a metric consists of establishing a mapping between the generic specification of the metric and the concepts that are represented in the software artifacts to be measured [20]. The possibility of having several operationalizations allows the most appropriate formula to be selected (by considering the availability of raw data in a specific platform). This model contains the actual parameters and instructions that are inherent to the platform, which can be retrieved by using different methods (e.g., agents, APIs, platform tools, libraries). Figure 3 shows an excerpt from the SaaS quality metamodel in which different operationalizations of a metric are specified.

The Runtime Quality Model (see Figure 2 (3)), which is a model@run.time, specifies all of the monitoring requirements, metrics, calculation formulas, and configurations that are needed to access the services to be monitored during their execution.

Figure 2 (4) shows the Middleware Configurator, which is an important component of the architecture. This component allows the definition of the Runtime Quality Model by matching the metrics that are specified in the Monitoring Requirements Model with the platform-dependent formulas that are contained in the SaaS Quality Model.

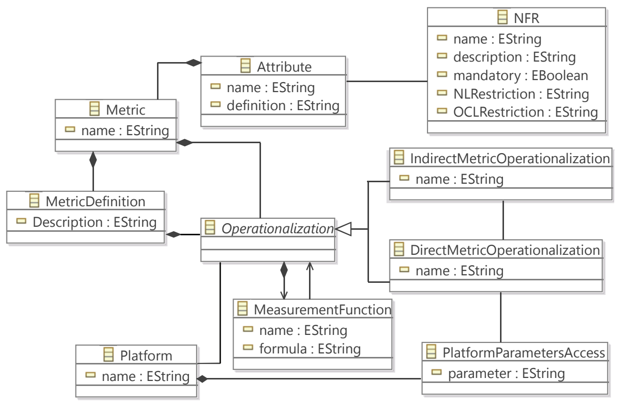


Figure 3. SaaS Quality Model Excerpt

Finally, Figure 2 (5) shows the Monitoring & Analysis Middleware containing two elements: the Measurements Engine, which calculates the monitoring values; and the Analysis Engine, which compares the calculated values with the thresholds that are specified in the Monitoring Requirements Model in order to be able to generate the SLA Violations Report. Since this approach has been proposed as a middleware, it will be implemented and deployed as a service in order to interact with the cloud services to be monitored. Moreover, it is possible to use ways to gather information from services deployed on any platform by mean of wrappers that allow the extension of a service to provide quality information. The main benefits of this architecture are its ability to monitor application-specific quality requirements and the flexibility and maintainability of the Monitoring & Analysis Middleware 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. Another potential benefit is its ability to detect faults before they occur.

V. INSTANTIATING THE MONITORING MIDDLEWARE IN MICROSOFT AZURE ©

In this section, we show how the platform-independent monitoring middleware can be instantiated. Despite the fact that the design and implementation of the middleware has been applied to monitor cloud services that are deployed on Microsoft Azure, similar actions can be taken to apply this solution to monitor cloud services that are deployed on other platforms like Amazon Web Services or Google App Engine.

In our monitoring architecture, the Measurements Engine must gather and calculate data regarding the behavior and performance of the cloud services, while the Analysis Engine determines whether the SLA and other NFRs for the services of interest are fulfilled. Figure 4 shows the Measurements Engine with three possible data-gathering scenarios. Each scenario is numbered and colored in the picture to show its context of use. The first scenario (1) shows the case in which raw data is gathered directly. This direct gathering of raw data can be achieved by 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 Performance Counters (provided by the Microsoft Azure Diagnostics Service). This service is an extension service that provides support for data extraction for roles and virtual machines that are deployed on Azure Platforms [21]. The Microsoft Azure Diagnostics Module 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-EC2, performance counters can be obtained from the Performance Monitor Counter [22] or from other mechanisms, depending on how each platform retrieves performance counters from cloud services.

The second scenario (2) is the case in which there are no Performance Counters for direct use. It is therefore 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 result can be managed by the Microsoft Azure Diagnostics Service. This scenario represents indirect metrics in quality terms. It can be applied in any platform since monitoring data is a result of measurements that are calculated from data that is gathered using scenarios (1) and (3).

Finally, the third scenario (3) is the case in which 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. This can be done by means of wrappers that encapsulate the corresponding cloud service. The mechanism used to produce the data needed from the service is hard coded in these wrappers, which can also be considered to be Custom Performance Counters, and store the gathered data in any storage solution. Here, we have used Azure Storage.

The following shows the implementation of the Monitoring & Analysis Middleware for the Microsoft Azure platform shown in Figure 5, which is based on the architecture presented in Figure 2.

First, the Middleware Configurator allows the service monitoring directives to be prepared. It is assumed that the SLA has been defined for the service to be monitored. The NFRs are contained in the Monitoring Requirements Model. These NFRs can be seen as a subset of the NFR categorization contained in the SaaS Quality Model. The SaaS Quality Model includes associated parameters that are specific to the platform to facilitate the selection of the appropriate metrics depending on the platform. The user must therefore use the Middleware Configurator to match the NFRs with the Azure Performance Counters (see Figure 6) (i.e., Azure Performance Counters, Custom Performance Counters, and Wrappers defined as Custom Performance Counters). Finally, the Middleware Configurator is able to use this information to generate the Runtime Quality Model. When the Runtime Quality Model has been generated, it is used by the Monitoring Middleware. In this example, we provide a detailed description of how we have developed both the Configurator and the Monitoring & Analysis Middleware.

The Middleware Configurator (shown at the top of Figure 5), was implemented in C# as an Azure Web Role by using Microsoft Visual Studio 2014. This configurator uses the Monitoring Requirements Model to specify the NFRs to be monitored. The specified NFRs must be matched with the formula that allows raw data to be collected from the services. The SaaS Quality Model is used to support the matching.

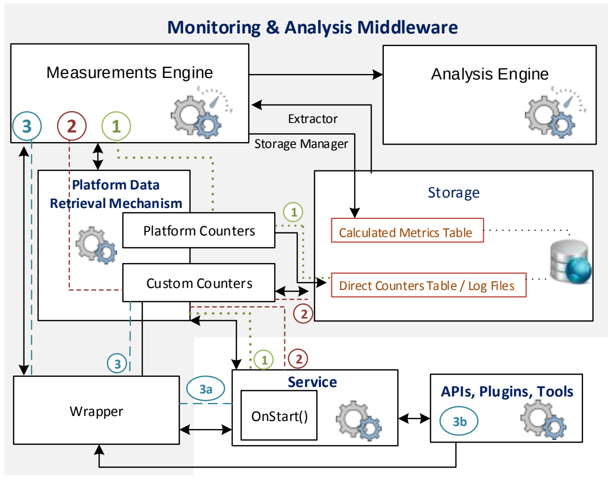


Figure 4. Platform-Independent Raw Data Extraction Scenarios

The Azure Service Counters must be matched with the NFRs that are contained in the Monitoring Requirements Model. When the matching is made, it is able to build the Runtime Quality Model. This model is consumed by the Measurements Engine allocated in the Monitoring & Analysis Middleware.

The Monitoring & Analysis Middleware begins to gather the raw data with the quality information from the services using one of the three 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 WadPerformanceCounters Table where the raw data is gathered by the Diagnostics Service directly from the services; and (2) the CalculatedMetrics Table, which contains the calculated metrics that are generated by the Measurements Engine and passed to the Analysis Engine.

It is also necessary to specify the sampling frequency of each Performance Counter, which may be different depending on the stakeholders settings, and the period of time in which the data is transferred to storage. Since the Runtime Quality Model can be changed, our example is able to handle updates of it at runtime, thus providing the monitoring infrastructure with flexibility.

The Extractor class has been implemented in the Monitoring & Analysis Middleware to retrieve data from the tables to operate with it. Thus Extractor class (Figure 5) retrieves raw data in each period. Since Microsoft Azure bills per transaction, it is beneficial to retrieve all the necessary information in the fewest transactions possible. Therefore our solution saves on costs to the stakeholders.
The Performance Counters provide two kinds of values: (i) counters, which specify a single value of a single data retrieval (e.g., the Request Execution Time of the last request); or (ii) accumulative data (e.g., the Total Number of Requests handled by the Service since it started). The client should therefore use the configurator to specify how to give meaning to the data retrieved in order to provide the highest quality measurements.

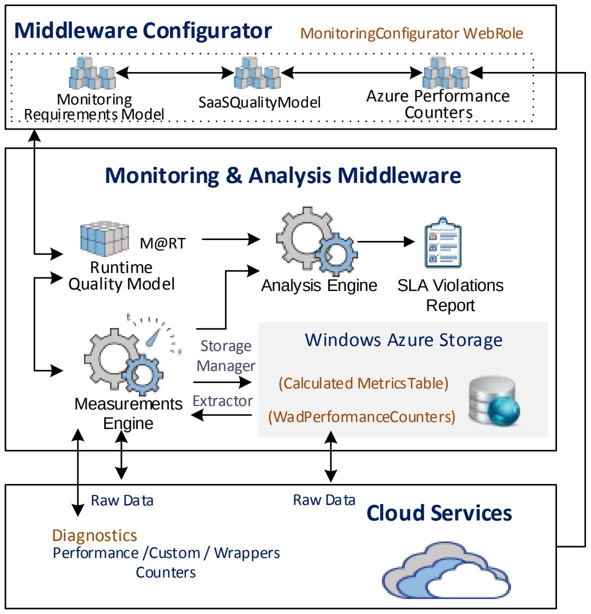


Figure 5. Monitoring Solution for the Azure Platform

Once the metrics have been calculated by the Measurements Engine, these values are stored in the Calculated Metrics Table (See Figure 5) to be analyzed by the Analysis Engine. The decision to use Windows Storage Tables was made because Windows Azure encourages the use of this kind of database, which is the most economic since it uses basic operations to store and retrieve data. However, the storage process is completely extensible to other databases if necessary.

The monitoring takes place by default at the service level, which means that metrics represent the behavior of the set of roles that make up the service. Moreover, by using this proposal it is possible to maintain the measure of the metrics at the role level. This occurs if the stakeholder is interested in obtaining in-depth knowledge of the issues in order to find solutions (e.g., several roles performing at an excellent level could conceal one that is performing poorly, as the general results will be positive). However, although sometimes appropriate, this option will imply higher storage and computational costs. It is up to the user to accept the tradeoff because the Monitoring Middleware allows both possibilities.

The monitoring middleware application has been exemplified through the use of a representative domain in which NFRs for SaaS architectures in cloud environments can be monitored and analyzed [23]. This example has been selected since it has been proposed for SaaS, thus permitting relevant NFRs to be monitored in cloud environments. The example is denominated as the Open Reference Case (ORC), which is used as an open source demonstrator to highlight the achievements of the European research project SLA@SOI. The ORC is an extension of the CoCoMe implementation [24] and provides a service-oriented retail solution that can be used in a trading system to handle the sales and stocking process [23]. For this running example, we have chosen Availability. We selected Availability because it is a relevant quality characteristic in cloud computing [25] and it clearly illustrates the use of the monitoring middleware when using both Azure Performance Counters and Custom Performance Counters.

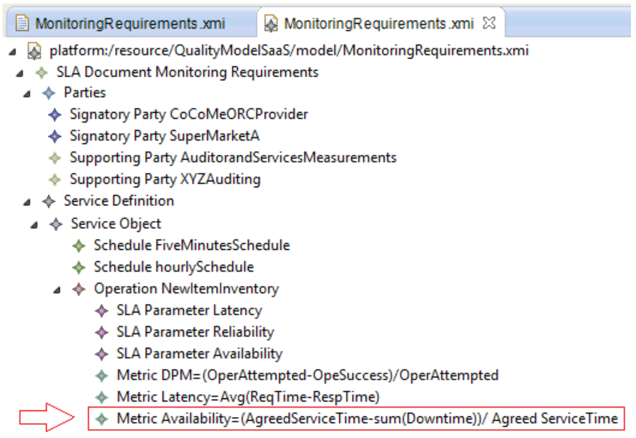
One of the SLA terms contains a commitment to making the inventory service in the retail solution available 99.50% or more of the time in a given calendar month. If the service offered fails to meet this commitment, a service credit will be applied to the customer's account. Availability is defined as the "ability of an IT service or other configuration item to perform its agreed function when required" [26]. Availability is mathematically expressed in (1), while the availability formula is defined in [27]:

$$\text{Availability} = \frac{\text{Agreed Service Time - Outage Downtime}}{\text{Agreed Service Time}} \qquad (1)$$

Agreed Service Time is the period during which the inventory service should be enabled. If downtime is permitted, the planned and scheduled downtime can be excluded from the Service Agreed Time. Outage Downtime is defined as "the sum, over a given period, of the weighted minutes, a given population of a systems, network elements, or service entities" [27]. It is sometimes necessary to provide different users with several different functions simultaneously, meaning that partial capacity and partial functionality outages are often more common than the total outages [28]. Prorating is achieved by using a portion of the capacity of the primary functionality that is impacted. In this example, we assume that the Outage Downtime of a Web or Worker Role in Azure affects the entire service provision, such that if a role fails, the service fails.

The first step for monitoring a given service is to identify the NFR to be monitored. The Monitoring Requirements Model with the NFR to be monitored is therefore the first input of the Monitoring Configurator. Figure 6 shows an excerpt of the model in question.

The SaaS Quality Model represents an important and necessary support for establishing the metrics of the requirements model. An excerpt from the meta-model, which contains the core of the SaaS Quality Model, is shown in Figure 3. However, the SaaS Quality Model itself is not shown due to space limitations. The SaaS Quality Model contains the metrics at a high level, along with the specific platform parameters that are used to obtain the measurement results. It therefore represents the next input for the Monitoring Configurator.


MonitoringRequirements.xmi | MonitoringRequirements.xmi

```
platform:/resource/QualityModelSaaS/model/MonitoringRequirements.xmi
  SLA Document Monitoring Requirements
    Parties
      Signatory Party CoCoMeORCProvider
      Signatory Party SuperMarketA
      Supporting Party AuditorandServicesMeasurements
      Supporting Party XYZAuditing
    Service Definition
      Service Object
        Schedule FiveMinutesSchedule
        Schedule hourlySchedule
        Operation NewItemInventory
          SLA Parameter Latency
          SLA Parameter Reliability
          SLA Parameter Availability
          Metric DPM=(OperAttempted-OpeSuccess)/OperAttempted
          Metric Latency=Avg(ReqTime-RespTime)
          Metric Availability=(AgreedServiceTime-sum(Downtime))/ Agreed ServiceTime
```

Figure 6. Requirements Quality Model Instance

As an example, the SaaS Quality Model contains two operationalizations of Availability (2) (3). These operationalizations are independent of the platform.

$$\text{Availability} = \frac{\text{Uptime}}{\text{Agreed Service Time}} \qquad (2)$$

The Azure Performance Counters can be employed to directly calculate the Operationalization of Availability by using (3), bearing in mind that the Outage Downtime is the sum of each period in which the service has not been available.

In order to calculate Availability, the Agreed Service Time is the initial period from which Availability must be guaranteed and specified in the SLA until the report date. However, all the historical data can be retrieved for the future reporting of the stakeholders' needs. When the formula used to capture raw data is set as in (4) and the matching between (3) and the formula using the Agreed Service Time minus the sum of (4) between a period has been employed, the configurator is able to generate the model@run.time, which in our approach is denominated as the Runtime Quality Model. This model is consumed by the middleware in order to capture the raw data from the cloud services and to apply the metrics.

$$\text{Availability} = \frac{(\text{Agreed Service Time} - \Sigma\ \text{Downtime})}{\text{Agreed Service Time}} \tag{3}$$

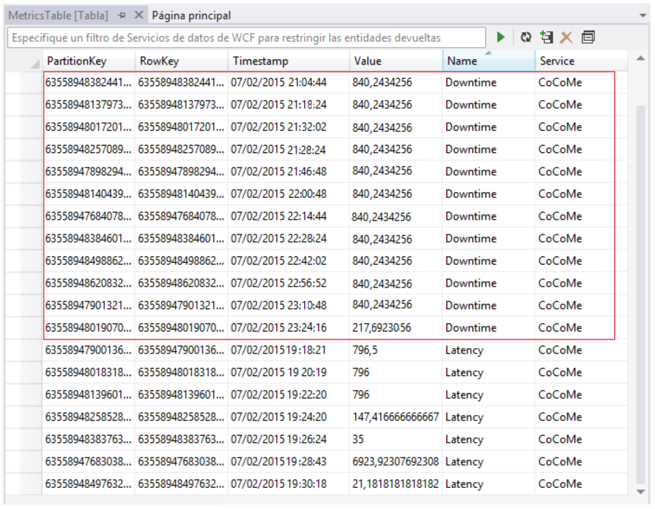
The SLA violations regarding availability are reported by storing Downtime in the Calculated Metrics table. Downtime is calculated by using the Azure Performance Counter "\Web Service(_Total)\ Service Uptime". This counter represents a cumulative value, which stores the service uptime. If the service is restarted, the uptime is a negative value. Thus, if ServiceUptimePreciseTimeStamp1 > ServiceUptimePrecise TimeStamp2, then downtime is calculated by using (4).

$$\text{Downtime} = (\text{Uptime2.PreciseTimeStamp} - \text{Uptime1.PreciseTimeStamp}) - \text{Uptime 2} \tag{4}$$

Downtime and Availability represent extraction scenario (2) since they are Custom Performance Counters. Downtime values are calculated using the uptime Azure Performance Counter. Moreover, we are monitoring cloud services at the SaaS level, so the availability is measured at that level; if the platform stops working, no data will be recorded by the middleware. Therefore, the last monitoring timestamp before the interruption and the first timestamp after the interruption are considered to measure the availability. Exceptions of this kind can be solved by the second scenario, using the same formula (4). The Downtime entries must be added in a determinate period. An excerpt from the Calculated Metrics Table is shown in Figure 7. This table shows the calculated downtime. From the downtime, availability is calculated. We have also monitored the latency, which can be calculated following the same steps. The results are presented in Table 7. The NFR that has been calculated using the Calculated Metrics Table allows the Analysis Engine to compare the expected values with the real values and the NFR Violations Report can then be generated.

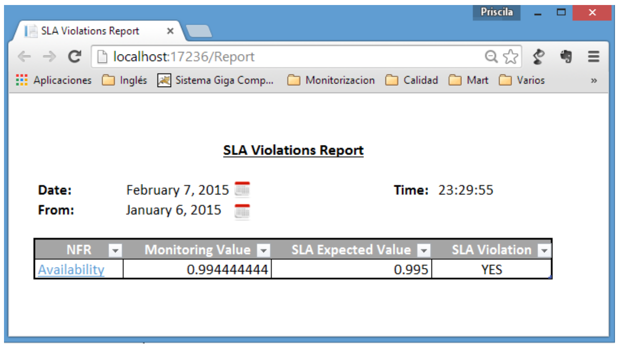
Figure 8 shows the SLA violations report, which includes the monitoring results. In this case, if the SLA requires an availability of 99.50% in a period between two timestamps and the Measurements Engine of the Monitoring & Analysis Middleware uses the monitoring results to determine that the real availability of a service was 99.444%, there has been an SLA violation. The Measurements Engine calculates the metric by using the sum of the Downtime values (shown in Figure 7), and applies formula (3). Therefore, the Analysis Engine performs the comparisons and determines whether an SLA violation has occurred.

Finally, it is important to bear in mind that the Monitoring Middleware can be developed for any platform by using the proposed architecture. The way in which the data will be retrieved from services is a key aspect to be considered. We have presented three different ways in which to obtain raw data on Azure Platforms. It is important to explore tools that are suitable for capturing data depending on the platform. The Diagnostic service has been used in this solution, although each platform presents APIs, tools, or libraries with which to capture raw data. If no way is established, the definition of Custom Performance Counters is always a valid option.


MetricsTable [Tabla]
Página principal
Especifique un filtro de Servicios de datos de WCF para restringir las entidades devueltas

PartitionKey	RowKey	Timestamp	Value	Name	Service
63558948382441...	63558948382441...	07/02/2015 21:04:44	840,2434256	Downtime	CoCoMe
63558948137973...	63558948137973...	07/02/2015 21:18:24	840,2434256	Downtime	CoCoMe
63558948017201...	63558948017201...	07/02/2015 21:32:02	840,2434256	Downtime	CoCoMe
63558948257089...	63558948257089...	07/02/2015 21:28:24	840,2434256	Downtime	CoCoMe
63558947898294...	63558947898294...	07/02/2015 21:46:48	840,2434256	Downtime	CoCoMe
63558948140439...	63558948140439...	07/02/2015 22:00:48	840,2434256	Downtime	CoCoMe
63558947684078...	63558947684078...	07/02/2015 22:14:44	840,2434256	Downtime	CoCoMe
63558948384601...	63558948384601...	07/02/2015 22:28:24	840,2434256	Downtime	CoCoMe
63558948498862...	63558948498862...	07/02/2015 22:42:02	840,2434256	Downtime	CoCoMe
63558948620832...	63558948620832...	07/02/2015 22:56:52	840,2434256	Downtime	CoCoMe
63558947901321...	63558947901321...	07/02/2015 23:10:48	840,2434256	Downtime	CoCoMe
63558948019070...	63558948019070...	07/02/2015 23:24:16	217,6923056	Downtime	CoCoMe
63558947900136...	63558947900136...	07/02/2015 19:18:21	796,5	Latency	CoCoMe
63558948018318...	63558948018318...	07/02/2015 19 20:19	796	Latency	CoCoMe
63558948139601...	63558948139601...	07/02/2015 19:22:20	796	Latency	CoCoMe
63558948258528...	63558948258528...	07/02/2015 19:24:20	147,416666666667	Latency	CoCoMe
63558948383763...	63558948383763...	07/02/2015 19:26:24	35	Latency	CoCoMe
63558947683038...	63558947683038...	07/02/2015 19:28:43	6923,92307692308	Latency	CoCoMe
63558948497632...	63558948497632...	07/02/2015 19:30:18	21,1818181818182	Latency	CoCoMe

Figure 7. Calculated Metrics Table



SLA Violations Report

Date: February 7, 2015 **Time:** 23:29:55
From: January 6, 2015

NFR	Monitoring Value	SLA Expected Value	SLA Violation
Availability	0.994444444	0.995	YES

Figure 8. SLA Violations Report Excerpt

VI. Conclusions

This paper presented a platform-independent monitoring middleware for cloud services that supports the monitoring of SLA compliance and provides a report containing SLA violations. This report may help service providers to understand the quality of the services with which their customers are provided and to make decisions regarding how to improve the quality of cloud services. In addition, we implemented the proposed middleware by using a runtime quality model. The application of models@run.time provides our solution with a high level of flexibility and maintainability; because changes essentially affect the runtime quality model rather than the entire infrastructure. We have instantiated our solution in the Azure Platform in order to be able to monitor any cloud service running in Azure. We have also defined three scenarios in which raw data is captured from the services: using Diagnostics Azure Service and its Performance Counters; using Diagnostics Azure Service and creating customized counters; and using "wrappers" to support customized information that cannot be captured by using the Azure Diagnostics Tools.

Our current efforts focus on improving the usability of the monitoring middleware by providing consumers with facilities that can be used to specify the way in which a report should be viewed (e.g., chart or graph) and other facilities that will allow the monitoring results to be shared with others and the monitored historical data to be maintained. In addition, we are exploring the use of new techniques to be able to extract more valuable data from cloud services or to be able to cope with more complex quality attributes (e.g., security).

As future work, we plan to perform an empirical evaluation of the efficiency, effectiveness, perceived ease of use, and perceived usefulness of end-users applying the monitoring middleware on different cloud platforms and with as many NFRs as possible. Furthermore, our approach does not yet reason about the interplay of service quality parameters and SLA requirements across multiple layers (software as a service, platform as a service, and infrastructure as a service). We plan to address this concern in future work. We also plan to explore the use of the monitoring middleware to detect faults in cloud services before they occur. This may imply the correlation of events with data from different sources (e.g., social media) in order to predict how external events can impact the quality of cloud services.

ACKNOWLEDGMENT

This research is supported by the Value@Cloud project (TIN2013-46300-R); the Scholarship Program Senescyt - Ecuador; University of Cuenca – Ecuador; and the Microsoft Azure for Research Award Program.

REFERENCES

[1] W. Tsai, X. Bai, and Y. Huang, "Software-as-a-service (SaaS): perspectives and challenges", Sci. China Inf. Sci., vol. 57, no. 5, 2014, pp. 1–15, doi: 10.1007/s11432-013-5050-z.

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

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

[4] C. Muller, M. Oriol, X. Franch, J. Marco, M. Resinas, A. Ruiz-Cortes, and M. Rodriguez, "Comprehensive Explanation of SLA Violations at Runtime", *Serv. Comput. IEEE Trans.*, vol. 7, no. 2, 2014, pp. 168–183, doi: 10.1109/TSC.2013.45.

[5] G. Katsaros, G. Kousiouris, S. V. Gogouvitis, D. Kyriazis, A. Menychtas, and T. Varvarigou, "A Self-adaptive hierarchical monitoring mechanism for Clouds", *J. Syst. Softw.*, vol. 85, 2012, pp. 1029–1041, doi: 10.1016/j.jss.2011.11.1043.

[6] N. Bencomo, J. Whittle, P. Sawyer, A. Finkelstein, and E. Letier, "Requirements reflection: requirements as runtime entities", in *32nd Int.Conf. on Software Engineering*, 2010, vol. 2, pp. 199–202, doi: 10.1145/1810295.1810329.

[7] L. Baresi and C. Ghezzi, "The Disappearing Boundary Between Development-time and Run-time", in *Workshop on Future of Software Engineering Research FSE/SDP*, 2010, pp. 17–22, doi: 10.1145/1882362.1882367.

[8] N. Bencomo, R. France, B. H. C. Cheng, U. Aßmann, B. H.C. Cheng, and U. Abmann, *Models@run.time. Foundations, Applications and Roadmaps*. London: Springer Cham Heidelberg New York Dordrecht London, 2014, p. 318, doi: 10.1007/978-3-319-08915-7.

[9] G. Blair, N. Bencomo, and R. B. France, "Models@ run.time", *Computer (Long. Beach. Calif).*, vol. 42, no. 10, Oct. 2009, pp. 22–27, doi: 10.1109/MC.2009.326.

[10] P. Cedillo, J. Gonzalez-Huerta, E. Insfrán, and S. Abrahao, "Towards Monitoring Cloud Services Using Models@run.time", in *9th Workshop on Models@run.time co-located with17th Int.Conf. on MDE Languages and Systems MODELS 2014*, 2014, pp. 31–40.

[11] 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*, 2014, pp. 1–21, doi: 10.1007/s00607-014-0398-5.

[12] K. Fatema, V. C. Emeakaroha, P. D. Healy, J. P. Morrison, and T. Lynn, "A survey of Cloud monitoring tools: Taxonomy, capabilities and objectives", *Journal of Parallel and Distributed Computing*, vol. 74. pp. 2918–2933, 2014, doi: 10.1016/j.jpdc.2014.06.007.

[13] A. Keller and H. Ludwig, "The WSLA Framework: Specifying and Monitoring Service Level Agreements for Web Services", *J. Netw. Syst. Manag.*, vol. 11, no. 1, 2003, pp. 57–81, doi: 10.1023/A:1022445108617.

[14] 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 *36th Annual Computer Software and Applications Conference (COMPSAC)*, 2012, pp. 499–508, doi: 10.1109/COMPSAC.2012.68.

[15] J. Shao, H. Wei, Q. Wang, and H. Mei, "A Runtime Model Based Monitoring Approach for Cloud", in *IEEE 3rd International Conference on Cloud Computing (CLOUD)*, 2010, pp. 313–320, doi: 10.1109/CLOUD.2010.31.

[16] H. Ludwig and A. Keller, "Web Service Level Agreement (WSLA) Language Specification", 2003, pp. 1–110.

[17] ISO/IEC, "ISO/IEC 25010 Systems and software engineering - Systems and Software Quality Requirements and Evaluation (SQuaRE) -- System and software quality models.". 2011.

[18] (OMG) Object Management Group, "Structured Metrics Metamodel (SMM)", 2012.

[19] M. Ferreira, F. Ruiz, F. Manuel, C. Calero, A. Vallecillo, M. Piattini, B. Mora, F. García, F. Ruiz, M. F. Bertoa, C. Calero, A. Vallecillo, M. Piattini, and B. Mora, "Medición del Software Ontología y Metamodelo", La Mancha, España, España, 2006.

[20] A. Fernandez, S. Abrahão, and E. Insfran, "A Web Usability Evaluation Process for Model-Driven Web Development", *23rd Int. Conf. Adv. Inf. Syst. Eng.*, 2011, pp. 108–122.

[21] "Collect Logging Data by Using Azure Diagnostics", *MSDN Library*, 2014. [Online]. Available: https://msdn.microsoft.com/en-us/library/azure/gg433048.aspx. [Accessed: 08-Feb-2015].

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

[23] P. Wieder, J. M. Butler, W. Theilmann, and R. Yahyapour, Eds., *Service Level Agreements for Cloud Computing*. New York, NY: Springer New York, 2011, p. 357, doi: 10.1007/978-1-4614-1614-2.

[24] 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", 2008, pp. 16–53.

[25] A. J. Gonzalez and B. E. Helvik, "System Management to Comply with SLA Availability Guarantees in Cloud Computing", in *4th Int. Conf. on Cloud Computing Technology and Science*, 2012, pp. 325–332, doi: 10.1109/CloudCom.2012.6427508.

[26] "ITIL Glossary and Abbreviations - English", 2011. [Online]. Available: https://www.axelos.com/Corporate/media/Files/Glossaries/ITIL_2011_Glossary_GB-v1-0.pdf. [Accessed: 06-Feb-2015].

[27] E. Bauer and R. Adams, *Service Quality of Cloud-Based Applications*, vol. 18. : John Wiley & Sons, Inc., 2013, p. 344.

[28] Quality Excellence for Suppliers of Telecommunications Forum (QuEST Forum)", TL9000 Quality Management System Measurements Hadbook 5.0, 2012. [Online]. Available: http://tl9000.org.