A Monitoring Infrastructure for the Quality Assessment of Cloud Services

Priscila Cedillo *icedillo@dsic.upv.es*
Department of Computer Systems and Computation, *Universitat Politècnica de València*
Valencia, Spain

Javier Gonzalez-Huerta *gonzalez_huerta.javier@uqam.ca*
Départment d'Informatique, Université du Québec à Montréal
Montreal, Canada

Silvia Abrahao *sabrahao@dsic.upv.es*
Department of Computer Systems and Computation, *Universitat Politècnica de València*
Valencia, Spain

Emilio Insfran *einsfran@dsic.upv.es*
Department of Computer Systems and Computation, *Universitat Politècnica de València*
Valencia, Spain

Abstract

Service Level Agreements (SLAs) specify the strict terms under which cloud services must be provided. The assessment of the quality of services being provided is critical for both clients and service providers. In this context, stakeholders must be capable of monitoring services delivered as Software as a Service (SaaS) at runtime and of reporting any eventual non-compliance with SLAs in a comprehensive and flexible manner. In this paper, we present the definition of an SLA compliance monitoring infrastructure, which is based on the use of models@run.time, its main components and artifacts, and the interactions among them. We place emphasis on the configuration of the artifacts that will enable the monitoring, and we present a prototype that can be used to perform this monitoring. The feasibility of our proposal is illustrated by means of a case study, which shows the use of the components and artifacts in the infrastructure and the configuration of a specific plan with which to monitor the services deployed on the Microsoft Azure© platform.

Keywords: Model Driven Engineering, Models@run.time, Quality Assessment, Cloud Services, Service Level Agreement, Software as a Service.

1. Introduction

Software-as-a-Service (SaaS) is a an emerging software deployment model that makes software available entirely through the use of a web browser, while hiding the details regarding where the software is hosted or its underlying architecture [1]. SaaS is increasingly being used by web-based applications owing the benefits it provides for both users and service providers [2]. The terms under which a SaaS application 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 the cloud provider offers to its customers [3] (e.g. ensuring the availability of a service at least 99.5% of the time). Service providers are becoming interested in monitoring cloud services in order to assess compliance with the SLA, thus avoiding possible penalizations and improving service quality [4]. On the customer side, service monitoring provides information and Key Performance Indicators (KPIs) that are useful in the decision-making process [5].

Traditional monitoring technologies are restricted to static and homogeneous environments, and cannot therefore be appropriately applied to cloud environments [6]. Cloud computing has led to the emergence of new issues, challenges, and needs as regards measuring quality (e.g. elasticity, scalability, adaptability, timeliness) [5]. Moreover, when compared with other distributed systems such as Grid Computing, the monitoring of a cloud is more complex because of the differences in both the trust model and the view of resources/services presented to the user [7], in addition to the presence of multiple layers and service paradigms [5]. Unfortunately, existing cloud and general purpose monitoring solutions have several limitations, as reported by Muller et al. [8]: the SLAs they support are not sufficiently expressive to model real-world scenarios. They couple the monitoring configuration with a given SLA specification, the explanations of the violations are difficult to understand and even potentially inaccurate, and some proposals either do not provide an architecture or the cohesion of their elements is low. Furthermore, it is important to have flexible quality monitoring infrastructures that will allow service providers to modify the non-functional requirements (NFRs) to be monitored, based on SLAs variations.

We believe that Model Driven Engineering (MDE) may be a solution as regards providing the flexibility required to monitor infrastructures. However, establishing all the NFRs to be monitored when designing the monitoring infrastructure is not always possible (e.g., owing to SLA renegotiations, the addition of new NFRs to be monitored, changes in the cloud platform). In this context, Baresi and Ghezzi [9] advocate that future software engineering research should be focused on providing software with intelligent support at runtime, thus breaking across the current rigid boundary between development-time and runtime. It is therefore necessary to define approaches that will allow cloud services to be monitored and will also permit the addition of new requirements or the modification of existing ones at runtime without interrupting the service execution. This challenge can be confronted by using models@run.time [9].

In a previous paper, we presented the definition of a monitoring process for cloud services by using models@run.time [10], in which we established the tasks involved in the monitoring process. In this paper, we extend that work by presenting the monitoring infrastructure that using models@run.time is able to: i) retrieve data from the cloud services during their execution; ii) calculate derived metrics based on these data; and iii) report any eventual SLA violations. The contribution of this paper is therefore the definition of a monitoring infrastructure, its main components (i.e. Monitoring Configurator and Monitoring Middleware) and the artifacts used by the Monitoring Configurator (i.e., quality meta-models with which to generate the Requirements Quality Model, the SaaS Quality Model and the Runtime Quality Model), along with the interactions among them. The feasibility of our proposal is illustrated by means of a case study, which shows the use of the components and artifacts involved in the infrastructure and the configuration of a specific monitoring plan for the Microsoft Azure© platform.

The paper is structured as follows: In Section 2, we discuss the existing solutions used to monitor cloud services. In Section 3, we present the monitoring infrastructure, its components and artifacts, in addition to describing the meta-models that support the definition and generation of the model@run.time. In Section 4, we present a case study performed to illustrate the feasibility of the proposed monitoring infrastructure, focusing on the monitoring configuration. Finally, in Section 5, we present our conclusions and discuss future work.

2. Related Work

Several studies whose aim has been to analyze the monitoring tools and approaches that are available (e.g., [10], [5]) and their weaknesses and needs have appeared over the last few years. Fatema et al. [10] report the results of a survey in which they analyze cloud and general purpose monitoring tools. They identify practical capabilities that an ideal monitoring tool should possess in order to fulfill the objectives of both cloud providers and customers in different cloud operational areas. They conclude that most general purpose monitoring tools were not designed with the cloud in mind, signifying that most monitoring capabilities (e.g.

multi-tenancy, scalability, non-intrusiveness) are improved using cloud based monitoring tools. However, one of the drawbacks of cloud monitoring tools is their portability. This reinforces the fact that many cloud specific monitoring tools are commercial and vendor dependent, which makes the tools less flexible and portable and means that their results are neither extensible nor comparable to other platforms. Aceto et al. [5] analyze and discuss the properties of a monitoring system for the cloud. They conclude that cloud monitoring tools should have quality characteristics (e.g., scalability, elasticity, adaptability) that will enable them to tackle the challenges that cloud monitoring implies. However, they also conclude that current solutions still require considerable effort if desirable characteristics are to be attained.

Many public cloud providers currently offer their customers the ability to monitor cloud services using the monitoring tools available for CPU, storage and network [11]. These tools are closely integrated with their own cloud solutions. They are only concerned with monitoring the quality of the service attributes for the hardware resources (CPU, storage, and network) and lack the ability to monitor application-specific QoS parameters and SLA requirements (i.e., latency, performance). In addition, the majority of these commercial tools (e.g., CloudWatch, LogicMonitor) are not sufficiently flexible to allow service providers to extend the QoS parameters provided in order to monitor the fulfillment of SLAs.

Various approaches have also been proposed in academic environments. For instance, Emeakaroha et al. [12] propose an application monitoring architecture named Cloud Application SLA violation Detection architecture (CASViD). This architecture monitors and detects SLA violations on the application layer, and includes tools for resource allocation, scheduling, and deployment. Although their approach provides a good solution, it does not have a flexible means to change the NFRs and metrics to be monitored at runtime. Katsaros et al. [13] present a monitoring system that facilitates on-the-fly self-configuration in terms of both the monitoring time and the monitoring parameters. They propose the use of scripts to collect data; however, they do not specify how NFRs are matched with raw data gathered from scripts and how they interact with cloud services. Müller et al. [8] designed and implemented SALMonADA, a service-based system with which to monitor and analyze SLAs in order to provide an explanation of violations. They describe SLAs using a Monitoring Management Document (MMD) to be consumed by the monitoring infrastructure; however, the platform does not support those users who wish to choose alternative means to measure quality requirements. Smit et al. [14] present and implement an architecture using stream processing to provide service monitoring. They emphasize that their infrastructure is intended be used to monitor hybrid clouds and two tiered cloud architectures working on streaming data. The possibility of gathering information therefore depends on the information that can be provided by other solutions. Montes et al. [15] propose a cloud monitoring taxonomy, which is used as the basis to define a layered cloud monitoring architecture. They implement GMonE, a general-purpose cloud monitoring tool, which is claimed to cover all aspects of cloud monitoring by specifically addressing the needs of modern cloud infrastructures. Similarly, Povedano-Molina et al. [16] propose DARGOS, a distributed architecture for resource management and monitoring in clouds, which ensures an accurate measurement of physical and virtual resources in the cloud in an attempt to keep overheads down. However, the latter two approaches confront the provision of only physical and virtual resources and do not emphasize the specific quality aspects of SaaS. In summary, to the best of our knowledge commercial tools are mostly tightly coupled with certain cloud platforms, support the monitoring of specific NFRs, and have pre-established low-level metrics; they are therefore not sufficiently versatile to support the modification of NFRs or the customization of their operationalizations[1] at runtime. There are other proposals that allow the verification of SLA compliance, but they are not sufficiently flexible to support different operationalizations needed according to the specific cloud platform involved.

[1] Operationalizing a measure consists of establishing a mapping between the generic description of the measure and the concepts that are represented in the software artifacts to be measured [28].

3. Monitoring Infrastructure

In this section, we present the *Monitoring Infrastructure* that has been designed to support the monitoring process defined in [17] (see Figure 1). This infrastructure allows: i) the specification and configuration of NFRs to be monitored; ii) an interaction with cloud services to assess their quality at runtime; iii) and the generation of reports containing any eventual SLA violations. In order to achieve these goals and provide the required degree of flexibility when defining NFR metrics, in addition to supporting different means to gather information from cloud services, we have defined a set of components and artifacts that conform to the monitoring infrastructure by using models@run.time.

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* to configure the monitoring of services and obtain the *Runtime Quality Model*. The *Monitoring & Analysis Middleware* uses this Runtime Quality Model and relies on two engines: the *Measurements Engine*, which permits cloud service monitoring through the use of the raw service quality data gathered from cloud services and takes the measurements, and the *Analysis Engine*, which compares the expected values with the monitored values and can generate the SLA violations report. The details of each process and artifact are detailed in the following subsections.

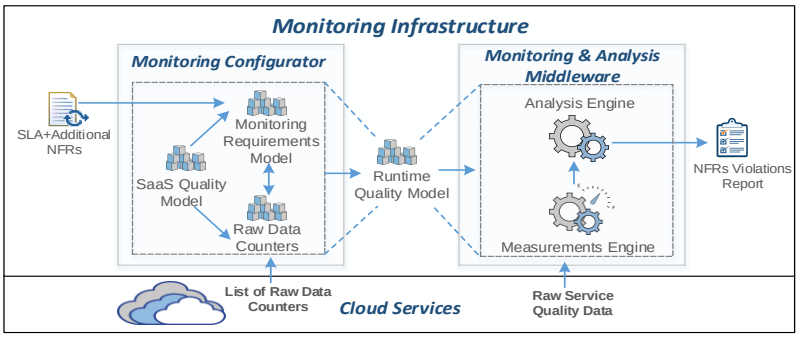


Fig. 1. Monitoring Infrastructure

3.1. Monitoring Configurator

The *Monitoring Configurator* is a component of the Monitoring Infrastructure (see Figure 1) and has a front-end which is used by stakeholders to configure the monitoring directives. It allows the high level NFRs to be monitored that are included in the *Monitoring Requirements Model* and the raw service quality data retrieved from cloud services to be matched. This matching is supported by the *SaaS Quality Model*, which acts as a guide that allows the selection of appropriate operationalizations for metrics. When the matching is done by stakeholders, the *Runtime Quality Model* is generated and can be consumed by the Monitoring & Analysis Middleware. A detailed description of the artifacts involved in the Monitoring Configurator and the interactions among them is shown below.

Monitoring Requirements Model

This model specifies the NFRs to be monitored in a way that can be comprehended by the Monitoring Infrastructure. It is compliant with the WSLA Language Specification [18] in order to represent NFRs in a standardized manner. Moreover, in our solution, the model is extended to support additional NFRs that are not part of SLAs but which may be of interest to stakeholders. Figure 2 shows the monitoring requirements meta-model, which incorporates all the SLA sections. The SLA specifies the parties, which are divided into signatory parties and supporting parties. On the one hand, signatory parties, namely service provider and service customer, are assumed to "sign" the SLA, while on the other, supporting parties are sponsored by signatory parties to provide service measurements and audits. The meta-model includes the *SLAParameter* meta-class, which represents the NFRs to be monitored and the *Metrics* used

to perform measurements. A *Service Object* is the abstraction of a service, whose quality characteristics and attributes are relevant as regards defining the SLA's terms. Characteristics and attributes are specified as *SLAParameters*. Each SLAParameter can be measured by using metrics. The SLAParameter meta-class has an attribute named *isSLATerm*, which differentiates an SLA term from an NFR that is not included in the SLA. The *Obligation* meta-class contains two types of obligations: i) a *Service Level Objective*, which is a guarantee of a particular state of SLA parameters in a given time period. (e.g. the average response time must be 5 ms) and ii) The *Action Guarantee*, which specifies the provider's commitment to doing something in a specific situation [21] (e.g. if a violation of a guarantee occurs, a notification is sent specifying a penalty). The values used as thresholds are obtained from the Action Guarantee meta-class (e.g. the response time must be less than 0.7 unless the transaction rate is greater than 1000). In this meta-model, a metric can be measured by using the formula agreed by the parties. A more detailed specification of the WSLA used to define the meta-classes, along with examples, can be found in [18].

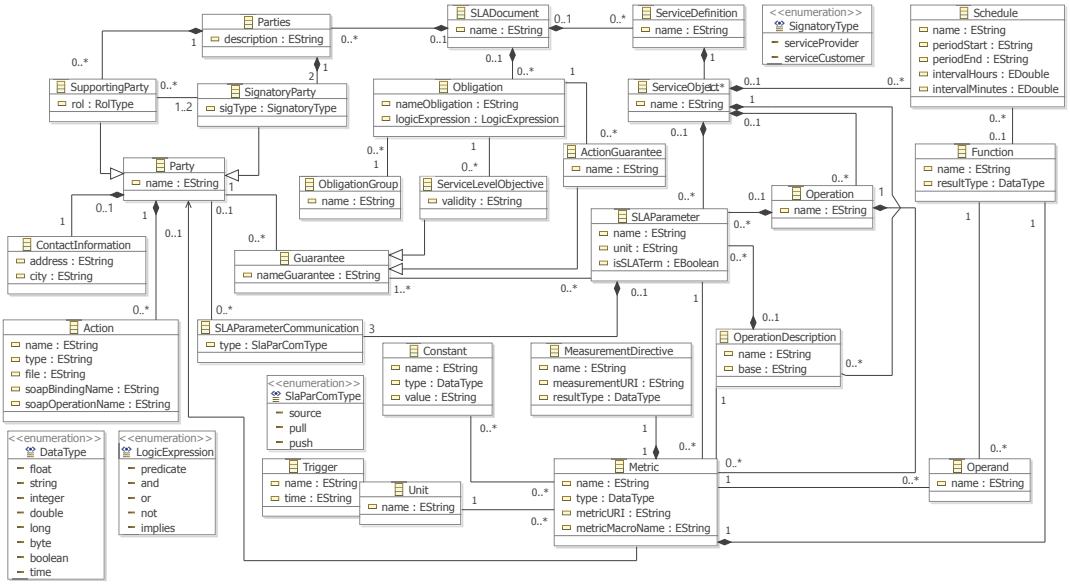


Fig. 2. Monitoring Requirements Meta-model

SaaS Quality Model

This model is aligned with the ISO/IEC 25010 standard (SQuaRE) [19]. Figure 3 shows the meta-model used to define the SaaS Quality Model. This model allows the definition of the whole set of *Characteristics*, *Sub-characteristics*, *Attributes*, their *Impact* (i.e., the relationships among attributes), and *Metrics* that specify how NFRs should be measured to assess the quality of cloud services. Each metric can be *operationalized* in different ways. A metric *Operationalization* can be considered at different *Cloud Levels* (i.e., SaaS, PaaS, IaaS). This is useful owing to the fact that there are a number of quality requirements (e.g., scalability, elasticity, security) that need to be monitored for different levels of service provision [5]. Moreover, it is important to specify the stakeholder that will use the monitoring information; for example, if the stakeholder is a service provider, it may be interesting to know the average number of users requesting a service at a particular time. The purpose of having *Perspectives* associated with each operationalization is to express whether a given operationalization is stakeholder-specific.

This information is useful during the processes of contrasting, improving measurements, or choosing different formulas with which to measure each NFR. The *DirectMetricOperationalization* meta-class represents a measure of an attribute that does not depend upon any other measure, whereas the *IndirectMetricOperationalization* meta-class represents measures that are derived from other *DirectMetricOperationalizations* or *IndirectMetricOperationalizations*. The *Platform* and *MeasurementMethod* meta-classes have been added to the SaaS Quality Model to maintain a list of raw data counters, which are platform dependent, and

because they facilitate the retrieval of information from a specific platform. Finally, the meta-model includes particularities of each operationalization, such as the *Unit* meta-class, which expresses the magnitude related to a particular quantity. The *Scale* meta-class represents a set of values with continuous or discrete properties that are used to map the operationalization.

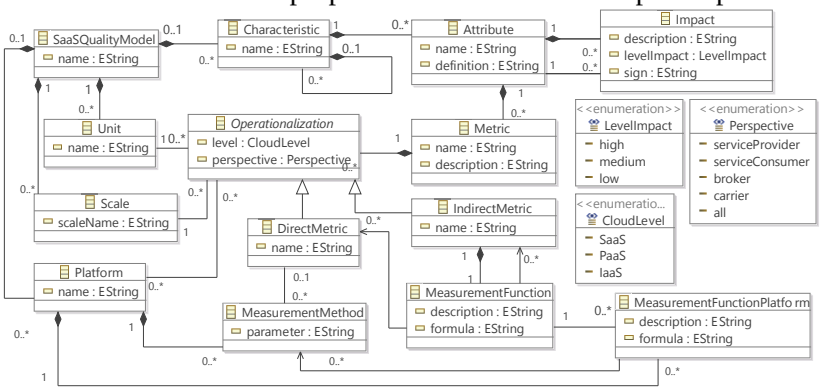


Fig. 3. SaaS Quality Meta-model

Runtime Quality Model

This is a model@run.time, which specifies the monitoring requirements, metrics, operationalizations, and configurations that will be used during the monitoring. Lehmann et al. [20] argue that meta-models at runtime must provide modeling constructs that will enable the definition of: (a) A prescriptive part of the model, specifying what the system should be like; (b) A descriptive part of the model specifying what the system is like; (c) Valid model modifications of the descriptive parts, executable at runtime; (d) Valid model modifications of the prescriptive parts, executable at runtime; (e) Causal connection, which is in the form of an information flow between the model and the entity being monitored. Figure 4 shows the Runtime Quality Meta-model, which is an extension of the SaaS Quality Model. It has many of the meta-classes included in the SaaS Quality Model described previously, plus meta-classes that represent the prescriptive part, the descriptive part, and the characteristics of the cloud platform that allow the causal connection.

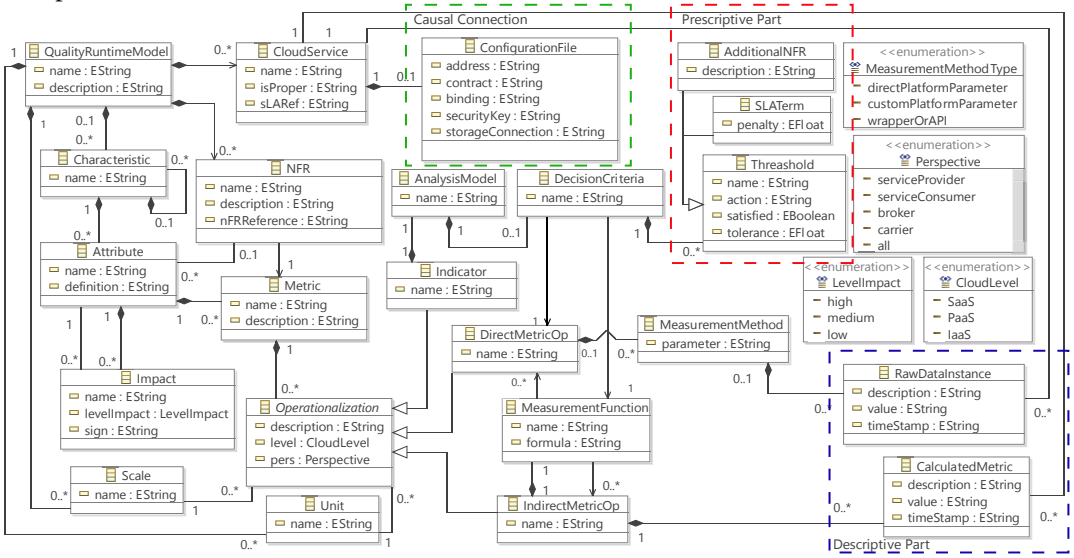


Fig. 4. Runtime Quality Meta-model

The *CloudService* meta-class also describes the service to be monitored. The prescriptive part of the model thus includes the *Threshold*, which can be a *SLATerm* threshold, obtained from the obligations part of the SLA, or an *AdditionalNFR* threshold set by the stakeholder. The descriptive part of the model is formed of the *RawDataInstance* meta-class, which contains the values captured directly from the cloud, and the *CalculatedMetric* meta-class,

which contains the measurement results of the calculated metrics. The ConfigurationFile meta-class therefore contains specific information for each platform that allows an interaction to take place between the monitoring infrastructure and the cloud service. It can therefore be considered as the class that is used to attain the causal connection between the monitoring infrastructure and services when a change needs to be reflected. Finally, the *Indicator* meta-class represents a measure that is derived from the other measures using an *Analysis Model* as a measurement approach [21]. In conclusion, the *Runtime Quality Model* allows our proposal to obtain the desirable characteristics related to flexibility and maintainability, since changes in the Runtime Quality Model can be easily reflected in the monitoring infrastructure.

Interaction Among Models

Figure 5 shows the interactions among the models. The first interaction (1) occurs between the SaaS Quality Model and the Monitoring Requirements Model. Stakeholders can use the SaaS Quality Model, which contains a standardized classification of characteristics, sub-characteristics, metrics, and attributes, as support in order to define the Monitoring Requirements Model. The second interaction (2) then occurs between the Monitoring Requirements Model and the Runtime Quality Model. Here, the stakeholder uses the Monitoring Configurator Interface to capture the NFRs and metrics included in the Monitoring Requirements Model to define the Runtime Quality Model. Finally, the third interaction (3) occurs between the Runtime Quality Model and the SaaS Quality Model. This interaction allows the means used to gather information from cloud services to be specified. In this scenario, the SaaS Quality Model is useful as regards matching the high level attributes contained in the Monitoring Requirements Model with raw service quality data. Here, the SaaS Quality Model enables a choice to be made from among many equivalent operationalizations with different measurement methods, thus providing our approach with flexibility. Once the interaction has been completed, the Runtime Quality Model can be used by the Monitoring & Analysis Middleware.

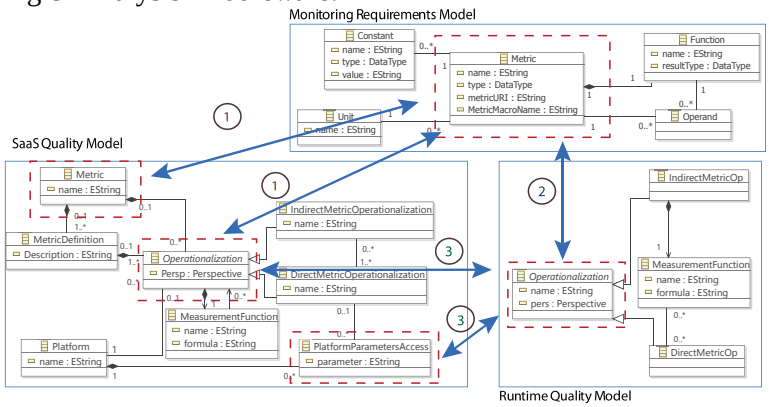


Fig. 5. Interaction between Models

3.2. Monitoring & Analysis Middleware

The Monitoring & Analysis Middleware consists of the Measurements Engine, which uses the Runtime Quality Model obtained as result of the configuration as input, and this applies metrics with which to measure the quality of services. There is also the Analysis Engine, which permits the analysis of quality and reports SLA violations. A detailed description of the Monitoring & Analysis Middleware components will be addressed in future work, since the scope of this paper is mainly focused on the monitoring configuration.

4. Case Study

An exploratory case study was performed following the guidelines presented in [22] in order to analyze the feasibility of the configuration task. The stages of the case study are: design, preparation, collection of data, and analysis of data, each of which is explained below.

4.1. Design of the Case Study

The case study was designed by considering the five components proposed in [22]: purpose of the study, underlying conceptual framework, research questions to be addressed, sampling strategy, and methods employed.

The **purpose** of this case study is to analyze the feasibility of configuring the monitoring of services by means of the Monitoring Configurator, and to use these configurations to generate the Runtime Quality Model. The Monitoring & Analysis Middleware will take this model as input to monitor the cloud services. The conceptual framework that links the phenomena to be studied is based on the Monitoring Process [17] and an infrastructure that supports this process (i.e., components, artifacts). The **research questions** to be addressed are: a) is the strategy of configuring and matching the NFRs with quality raw data retrieved from cloud services to obtain the desired monitoring information useable and effective?; b) what are the limitations of the monitoring configurator?

Here, the sampling strategy is based on monitoring configuration tests carried out by a subject who is an IT professional with programming skills and who has been working as a Cloud Provider Service Specialist for two years. In accordance with Lethbridge et. al [23], we have applied the second degree of data collection techniques, in which the researcher directly collects raw data without interacting with the subject during the data collection.

In order to collect the monitoring information, we have developed a prototype of the Monitoring & Analysis Middleware, which allows the collection of raw runtime data through the use of the Runtime Quality Model generated in the configuration task. The monitoring configuration was carried out as follows: the subject used the Monitoring Requirements Model to match NFRs with quality parameters and instructions that gather information from a service running in the cloud. The technique used to obtain feedback regarding the feasibility of the monitoring configuration performed was an analysis of the monitoring results obtained using a prototype of the Monitoring Engine in order to obtain the data needed to prove whether the values gathered were those expected by the subject.

4.2. Preparation of the Case Study

The context of this case study, was a test scenario in which the subject carried out the monitoring configuration. The SaaS Quality Model was used to support the matching between the NFRs to be monitored and the platform information. Once this information had been matched, it was possible to generate the Runtime Quality Model, which was then used by the Monitoring & Analysis Middleware to gather, measure and analyze quality data obtained from cloud services. The services used in this case study were implemented in compliance with an Open Reference Case (ORC) proposed in [18], which was 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], which provides a service oriented retail solution that can be used in a supermarket trading system to handle the sales and stocking process [25]. The set of services defined by ORC was deployed as a SaaS on the Microsoft Azure© platform. We considered the actions (i.e., create, read, update, and delete operations) related to the inventory service and the sales service. The objective was to configure the monitoring infrastructure in order to perform quality evaluations of cloud services. The NFRs to be monitored were reliability and latency.

Figure 6 presents an excerpt of an instance of the Monitoring Requirements. It shows the service, its operations (e.g NewItemInventory) and the NFRs (SLAParameters). The NFRs to be monitored are the *reliability* and *latency* of the inventory and sales cloud services. *Reliability* is defined as *"the ability of an item to perform a required function under stated conditions for a stated time period"* [26]. Customers and suppliers often measure service reliability as Defective operations Per Million attempts (DPM) [27]. In this case study, the SLA term included the following clause: "the service could have a maximum of ten defective operations per million" (i.e., "99.999% service reliability"). *Service latency* was, meanwhile, defined as *"the time that has elapsed between a request and the corresponding response"* [27], and thus "the maximum service latency is 130 ms". The *Monitoring Requirements*

Model includes the DPM metric which measures reliability. It was then necessary to select the DPM equivalent operationalization, which allows the measurement of the reliability NFR in cloud services deployed on the Microsoft Azure © platform.

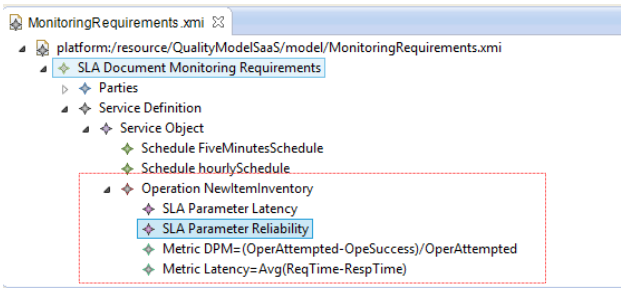


Fig. 6. Monitoring Requirements Model

Our *SaaS Quality Model* contains three equivalent metric operationalizations (i.e. DPM1, DPM2, and DPM3). The subject had to select one of them depending on the *Monitoring Requirements Model* and the Raw Service Quality Data enabled it to be retrieved from cloud services. The operationalizations included in our *SaaS Quality Model* to calculate DPM are:

$$\text{DPM 1} = \frac{\text{OperationsAttempted - Operations Successful}}{\text{Operations Attempted}} * 10^6 \quad (1)$$

$$\text{DPM 2} = \frac{\text{Operations Failed}}{\text{Operations Attempted}} * 10^6 \quad (2)$$

$$\text{DPM 2} = \frac{\text{Operations Failed}}{\text{OperationsSuccessful + OperationsFailed}} * 10^6 \quad (3)$$

The subject can select an equivalent operationalization by considering the advantages and disadvantages of the selection (e.g. overheads, ease of gathering information). Once the Runtime Quality Model has been generated, the Monitoring & Analysis Middleware is able to collect information, measure data, and report SLA violations. Here, data is captured by using the Azure Diagnostics Service. However, this could change depending on the facilities of each cloud platform. Diagnostics contains different counters with which to obtain data from cloud services. Here, the subject was able to use one of the three equations (1), (2), (3) to match that selection with Diagnostics counters. Finally, the matched formula was used for the Monitoring & Analysis Middleware using Diagnostics counters. We have developed the Monitoring Configurator, shown in Figure 7, which allows the monitoring configuration.

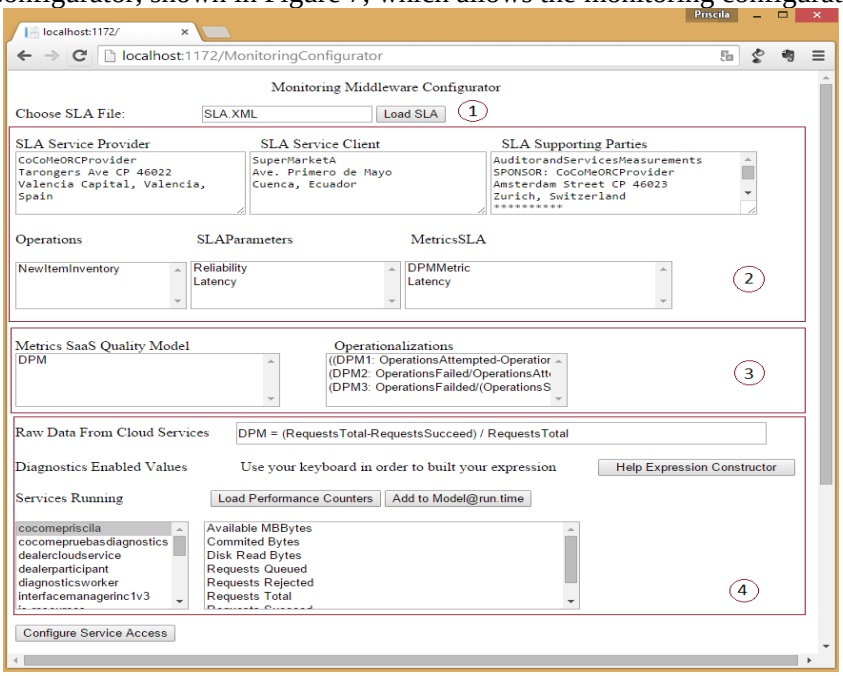
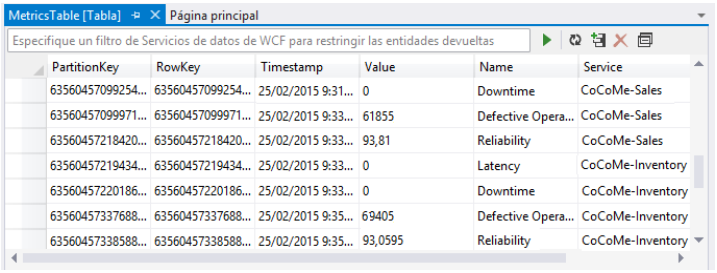


Fig. 7. Model@run.time Configurator Interface

4.3. Collection of Data

The data was collected in two stages: (1) when the subject carried out the configurations depending on the NFRs specified in the Monitoring Requirements Model and matched these NFRs with raw platform-specific data counters to generate the Runtime Quality Model using the SaaS Quality Model; (2) when the monitoring engine gathered and measured information provided by cloud services based on the Runtime Quality Model.

A prototype of the Monitoring & Analysis Middleware was implemented as a Microsoft Azure cloud service, which used the Runtime Quality Model to capture the raw data from the cloud, make measurements and store the results in a data base. Finally, Figure 8 shows the storage table containing the metrics calculated from the cloud service.


MetricsTable [Tabla] Página principal

Especifique un filtro de Servicios de datos de WCF para restringir las entidades devueltas

PartitionKey	RowKey	Timestamp	Value	Name	Service
63560457099254...	63560457099254...	25/02/2015 9:31...	0	Downtime	CoCoMe-Sales
63560457099971...	63560457099971...	25/02/2015 9:33...	61855	Defective Opera...	CoCoMe-Sales
63560457218420...	63560457218420...	25/02/2015 9:33...	93,81	Reliability	CoCoMe-Sales
63560457219434...	63560457219434...	25/02/2015 9:33...	0	Latency	CoCoMe-Inventory
63560457220186...	63560457220186...	25/02/2015 9:33...	0	Downtime	CoCoMe-Inventory
63560457337688...	63560457337688...	25/02/2015 9:35...	69405	Defective Opera...	CoCoMe-Inventory
63560457338588...	63560457338588...	25/02/2015 9:35...	93,0595	Reliability	CoCoMe-Inventory

Fig. 8. Metrics calculated by using the Monitoring Infrastructure

4.4. Analysis of Data

The monitoring configuration was analyzed so as to address our research questions. The subject used the Monitoring Requirements Model, which contained the NFRs to be monitored, and their metrics and thresholds. The subject then matched the metrics with the appropriate operationalizations specific to the platform. In order to illustrate the process used to monitor the reliability, the other NFRs were monitored following analogous steps. The reliability threshold was 99.999%, and we the considered operationalization (1) which was set up by matching formula (4) with the following Azure Counters:

- OperationsAttempted=@"\ASP.NET Applications(_Total_)\ Requests Total
- OperationsSuccessful=@"\ASP.NET Applications(_Total_)\ Requests Succeeded"

$$\mathrm{DPM} = \frac{\mathrm{RequestsTotal} - \mathrm{RequestsSucceded}}{\mathrm{RequestsTotal}} \quad (4)$$

The *RuntimeQuality Model* should therefore include Formula (4). When checking whether the monitoring infrastructure would be able to monitor the behavior of the cloud services by using the runtime quality model generated, we intentionally introduced exceptions into the ORC services' source code in order to generate problems as regards reliability and latency.

It was necessary to determine whether the configuration gathers the expected information from the cloud services by using the Runtime Quality Model and to find possible limitations or inaccurate results. Here, we have concluded that the Runtime Quality Model produced the expected values shown in the table presented in Figure 8, in which the exceptions introduced were reflected in the monitoring results (the reliability offered was 99.999% and the actual Reliability was 93.0595% for the inventory service, signifying that the SLA was violated).

Case Study Conclusions and Lessons Learned

With regard to the first research question stated for this case study, we provide support to help the configuration of NFRs to be monitored using our approach and that the configuration was effective as regards monitoring Azure cloud services. Moreover, the suitability of this approach is shown by the fact that it is feasible to use the Monitoring Configurator to match the NFRs included in the Monitoring Requirements Model with the raw service quality data gathered from the cloud service and provide the expected information. With regard to the second research question, the Monitoring Infrastructure is able to detect SLA violations from a wide range of NFRs. However, it is important to take into account that not all the NFRs can

be monitored owing to the restriction of the infrastructure that provides the raw service quality data from the services. One solution to this issue would be to use wrappers for services in order to capture the information required in a customized manner, which constitutes one of the next steps in our research.

As lessons learned this case study has allowed us to observe the potentialities and limitations of our proposal. The monitoring configurator allows a wide variety of operationalizations and platform counters to be matched. However, it depends on the facilities used to provide raw service quality data. During the execution of the case study, several aspects related to how the configuration can be facilitated have been discovered. For example, the SaaS Quality Model provides a simple means to choose the operationalizations and it is possible to add operationalizations to the SaaS Quality Model, which represents a knowledge base that saves efforts and minimizes possible mistakes when the configuring task is being carried out.

5. Conclusions and Future Work

In this paper, we have presented a monitoring infrastructure for cloud services, which allows data to be retrieved from cloud services in order to calculate monitoring metrics and eventually report non-compliance with the SLA. The monitoring infrastructure uses the Runtime Quality Model, which is generated by using two additional models: the Monitoring Requirements Model and the SaaS Quality Model. The feasibility of the approach has been illustrated by means of a case study which shows the monitoring of services deployed on the Azure platform.

The use of models@run.time provides flexibility and eases maintainability when the SLA and additional NFRs to be monitored change. Moreover, the facility of changing the model and not the monitoring infrastructure makes it easy for users to operate and understand in the case of their not being familiar with the middleware implementation.

As future work, we plan to deliver our Monitoring & Analysis Middleware in other platforms (e.g. Amazon AWS, Google) to be able to monitor and analyze services deployed in these platforms. We also plan to carry out a systematic review of the quality characteristics, sub-characteristics, attributes, and metrics of cloud services. The findings will be included in the SaaS Quality Model in order to study the monitoring mechanisms provided by other commonly used cloud platforms such as Google App Engine or Amazon AWS. Moreover, we plan to study generic means to encapsulate the raw data collected from the cloud services in order to obtain common interfaces for many platforms (e.g., APIs, proxies, plugins, wrappers). Finally, we plan to improve the efficiency of the proposal by taking in account issues such as overheads, security, etc. and to empirically validate the approach using controlled experiments.

Acknowledgments

This research is supported by the Value@Cloud project (TIN2013-46300-R), the ValI+D fellowship program (ACIF/2011/ 235), Scholarship Program Senescyt-Ecuador, University of Cuenca-Ecuador, NSERC (Natural Sciences and Engineering Research Council of Canada) and Microsoft Azure for Research Award Program.

References

[1] Sriram, I., Khajeh-Hosseini, A., "Research Agenda in Cloud Technologies," in *1st ACM Symposium on Cloud Computing, SOCC*, 2010, vol. cs.DC, pp. 1–11.

[2] Song, J., Han, F., Yan, Z., Liu, G., Zhu, Z., "A SaaSify tool for converting traditional web-based applications to SaaS application," *4th Int. Conf. CLOUD* pp. 396–403, 2011.

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

[4] Hassan, M., Song, B., Huh, E.-N., "A market-oriented dynamic collaborative cloud services platform," *Ann. Telecommun.*, vol. 65, no. 11–12, pp. 669–688, 2010.

[5] Aceto, G., Botta, A., de Donato, W., Pescapè, A., "Cloud monitoring: A survey,"

Comput. Networks, vol. 57, no. 9, pp. 2093–2115, Jun. 2013.

[6] Shao, J., Wei, H., Wang, Q., Mei, H., "A Runtime Model Based Monitoring Approach for Cloud," in *IEEE 3rd Int. Conf.on Cloud Computing (CLOUD)*, 2010, pp. 313–320.

[7] Foster, I., Zhao, Y., Raicu, I., Lu, S., "Cloud computing and grid computing 360-degree compared," in *Grid Computing Environments Workshop(GCE)*, 2008, pp. 1–10.

[8] Muller, C., Oriol, M., Franch, X., Marco, J., et al, "Comprehensive Explanation of SLA Violations at Runtime," *Serv. Com. IEEE Trans.*, vol. 7, no. 2, pp. 168–183, 2014.

[9] Baresi, L., Ghezzi, C., "The Disappearing Boundary Between Development-time and Run-time," *Workshop on Future of Soft. Eng. Research FSE/SDP*, 2010, pp. 17–22.

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

[11] Alhamazani, K., Ranjan, R., Mitra, K., Rabhi, F., Jayaraman, P. P., Khan, S. U., Guabtni, A., Bhatnagar, V., "An overview of the commercial cloud monitoring tools: research dimensions, design issues, and state-of-the-art," *Computing*, pp. 1–21, 2014.

[12] Emeakaroha, V. C., Ferreto, T. C., Netto, M. A. S., Brandic, I., De Rose, C. A. F., "CASViD: Application Level Monitoring for SLA Violation Detection in Clouds," in *36th Computer Software and Applications Conf. (COMPSAC)*, 2012, pp. 499–508.

[13] Katsaros, G., Kousiouris, G., Gogouvitis, S. V., Kyriazis, D., Menychtas, A., Varvarigou, T., "A Self-adaptive hierarchical monitoring mechanism for Clouds," *J. Syst. Softw.*, vol. 85, pp. 1029–1041, 2012.

[14] Smit, M., Simmons, B., Litoiu, M., "Distributed, application-level monitoring for heterogeneous clouds using stream processing," *Futur. Gener. Comput. Syst.*, vol. 29, no. 8, pp. 2103–2114, 2013.

[15] Montes, J., Sánchez, A., Memishi, B., et al., "GMonE: A complete approach to cloud monitoring," *Futur. Gener. Comput. Syst.*, vol. 29, no. 8, pp. 2026–2040, 2013.

[16] Povedano-Molina, J., Lopez-Vega, J. M., Lopez-Soler, J. M., Corradi, A., Foschini, L., "DARGOS: A highly adaptable and scalable monitoring architecture for multi-tenant Clouds," *Futur. Gener. Comput. Syst.*, vol. 29, no. 8, pp. 2041–2056, 2013.

[17] Cedillo, P., Gonzalez-Huerta, J., Insfrán, E., Abrahao, S., "Towards Monitoring Cloud Services Using Models@run.time," in *9th Workshop on Models@run.time*, 2014.

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

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

[20] Lehmann, G., Blumendorf, M., Trollmann, F., Albayrak, S., "Meta-modeling Runtime Models," in *Int. Conf. on Models in Software Engineering*, 2010, pp. 209–223.

[21] García, F., Bertoa, M. F., Calero, C., Vallecillo, A., Ruíz, F., Piattini, M., Genero, M., "Towards a consistent terminology for software measurement," *Information and Software Technology*, vol. 48. pp. 631–644, 2006.

[22] Runeson, P., Höst, M., "Guidelines for conducting and reporting case study research in software engineering," *Empir. Softw. Eng.*, vol. 14, no. 2, pp. 131–164, Dec. 2009.

[23] Lethbridge, T. C., Sim, S. E., Singer, J., "Studying software engineers: Data collection techniques for software field studies," *Empir. Softw. Eng.*, vol. 10, pp. 311–341, 2005.

[24] Herold, S., Klus, H., Welsch, Y., Deiters, C., Rausch, A., Reussner, R., Krogmann, K., Koziolek, H., Mirandola, R., Hummel, B., Meisinger, M., Pfaller, C., "CoCoME - The Common Component Modeling Example," 2008, pp. 16–53.

[25] Wieder, P., Butler, J. M., Theilmann, W., Yahyapour, R., Eds., *Service Level Agreements for Cloud Computing*. New York, NY: Springer New York, 2011.

[26] "Quality Excellence for Suppliers of Telecommunications Forum (Quest Forum), TL 9000 Quality Management System Measurements Handbook 5.0," 2012.

[27] Bauer, E., Adams, R., *Service Quality of Cloud-Based Applications*, vol. 18. Hoboken, New Jersey, USA: John Wiley & Sons, Inc., 2013.

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