Towards Monitoring Cloud Services Using Models@run.time

Priscila Cedillo, Javier Gonzalez-Huerta, Silvia Abrahao, Emilio Insfran

ISSI Research Group, Department of Information Systems and Computation
Universitat Politècnica de València,
Camino de Vera, s/n, 46022, Valencia, Spain
{icedillo, jagonzalez, sabrahao, einsfran}@dsic.upv.es

Abstract. Cloud computing represents a new trend to provide software services. In order to deliver these services there are certain quality levels that should be considered. The provided services need to comply with a set of contract terms and non-functional requirements specified by a service level agreement (SLA). In addition, to support the fulfillment of the SLA a monitoring process should be defined. This allows service providers to determine the actual quality level of services in the cloud. In this paper, we define a monitoring process for the usage of models at runtime, specifying low- and high-level non-functional requirements contained in a SLA. Models at runtime provide flexibility to the monitoring infrastructure due to their reflection mechanisms; the modification of non-functional requirements may dynamically change the monitoring computation, avoiding the need to adjust the monitoring infrastructure. In our approach, models at runtime are part of a monitoring middleware that interacts with cloud services; it retrieves data in the model at runtime, analyzes the information, and provides a report detailing the issues of non-compliance of non-functional requirements.

Keywords: Cloud Computing, SaaS, Models@run.time, SLA, Monitoring, Model Driven Engineering.

1 Introduction

The evolution of cloud computing technologies is promoting the development of new techniques to provide high-quality services. Cloud computing infrastructures, with software as a service model, provide capability to consumers to use software and services hosted in the cloud platform. Due to the nature of the cloud, the ways in which services are built and deployed have changed. As a result, it is necessary to fulfill non-functional requirements including the most specific characteristics of the cloud (e. g. scalability and elasticity).

Service Level Agreements (SLAs) emerge as a key aspect to ensure the expected quality level of the services between the consumer and the provider. ITIL defines a SLA as a formal, negotiated document in quantitative terms (and perhaps qualitative terms), detailing the service that will be offered to a customer [1]. Any metrics in-

cluded in a SLA should be capable of being measured on a regular basis and the SLA should record them [1]. Problems arise from the current practice in SLA specification for IT services because SLAs are mostly based on templates, mainly filled with natural language descriptions that make it difficult to automate SLA compliance verification [2]. In order to support the SLA fulfillment and timely reaction to failures, advanced SLA strategies are necessary. These techniques include appropriate resource-monitoring concepts. The Quality-of-Service (QoS) attributes, which are generally part of an SLA, change constantly in order to fulfill the agreement. As a result, these attributes need to be closely monitored [3].

Traditional monitoring technologies are restricted to static and homogenous environments and, therefore, cannot be appropriately applied to cloud environments [4]. In traditional software development, many assumptions in the context of an application are described at design time; however, in cloud computing, those assumptions are not possible [5]. Moreover, several non-functional assurance criteria may be more easily guaranteed at runtime than at design time. For example, it is easier to assess latency when it is possible to measure and continually monitor delay times in the running system [6]. Cloud computing brings new issues, challenges and needs in performance testing, evaluation and scalability measurements due to the special features of cloud computing, such as latency, elasticity and scalability [7]. The information of the system in execution feeds the models at runtime, which support reasoning, adaptation or monitoring of the system. To realize such a connection between the running system and the models at runtime, the system needs a self-representation of its quality view, which is used to map the raw data with the high-level requirements specified in the SLA.

Based on the utilization of models, a runtime model is defined as an abstraction of a running system which is being manipulated at runtime for a specific purpose [8]. Another definition of a model at runtime is a causally connected self-representation of the associated system that emphasizes the structure, behavior and goals of the system from a problem space perspective [9].

As far as we know, there is a lack of studies which uses models at runtime in cloud computing environments. Models at runtime are useful to support cloud services monitoring because developers do not need to implement new requirements that should be included for monitoring in the infrastructure; they only need to include them in the model. Moreover, cloud computing environments bring new issues and present particular characteristics that differentiate the ways in which we should measure their quality [10].

This paper presents an approach to monitor non-functional requirements of cloud services specified in the SLA using models at runtime, through a middleware that interacts with services or applications in the cloud. This middleware retrieves information from the running system and feeds the model at runtime, analyzing this information, and providing a report with issues that violate the SLA.

This approach is useful to measure higher-level attributes. It is important to consider that models at runtime give flexibility when the evaluator needs to change monitoring criteria or wants to change the parameters to be monitored; this is because the

monitoring system does not need to be adjusted in this case and only the attributes to be monitored over the model should be changed.

This work is structured as follows: In Section 2, we present related work addressing models at runtime and how they are used to monitor applications, SLA management, and quality requirement representations in SLAs. In Section 3, we present the monitoring process. In Section 4, we explain how the process works by means of an example. Finally, in Section 5, we present our conclusions and discuss future work.

2 Related Work

We classify related work into models at runtime and the way in which they are used to monitor applications. Since there is a lack of work focusing on monitoring using models at runtime in the cloud and web services, we look at other environments in which models at runtime are used in monitoring and which can represent a valid reference for this work [9][5,11,12,13,14,15]. Finally, we discuss the SLA management and quality requirements representations in the cloud [4][16,17,18, 19,20].

Baresi and Ghezzi [5] advocate that future software engineering research should focus on providing intelligent support to software at runtime, breaking today's rigid boundary between development-time and runtime. Szvetits et al. [11] build a classification and conduct a survey in terms of objectives, techniques, architectures and kinds of models using models at runtime. They observe the objectives pursued when using a system that utilizes models at runtime and conclude that one of the most important objectives is system monitoring. Bencomo et al. [12] show that models at runtime are an important research topic for *enterprise and cloud*, and included a session about this topic in the 8th International Workshop on Models@runtime. Bertolino et al. [13] propose a property-driven approach to runtime monitoring that is based on a metamodel and a generic configurable monitoring infrastructure; however, they do not pay attention to the particular characteristics of cloud computing (e.g. elasticity, scalability, etc). In [14], the authors develop the GLIMPSE monitoring infrastructure in the context of the European Project CONNECT that can support runtime performance analysis. Blair et al. [9] define models at runtime as being similar to a causally connected self-representation of the associated system that emphasizes the structure, behavior and goals of the system from a problem space perspective. Their vision of models at runtime is to raise the level of runtime model abstraction to that of requirements, and Bencomo et al. [15] use requirement reflection in self-adaptive systems by making requirements first-class runtime entities, thus endowing software systems with the ability to reason with, understand, explain and modify requirements at runtime.

Emeakaroha et al. [4] present a framework entitled LoM2HiS for the mapping of low-level resource metrics to high-level SLA parameters. Its architecture includes a runtime monitor that continuously monitors the customer's application status and performance; then in [16] they propose an application monitoring architecture entitled CASViD, which stands for Cloud Application SLA Violation Detection architecture. Correia et al. [17] propose a domain specific language (SLA Language for specifica-

tion and Monitoring – SLALOM) for SOA, in order to bridge the gap between the customer perspective (business oriented) and the service provider (implementation oriented, which becomes more evident in a SLA monitoring process). Myerson [18] discusses some best practices and how SLAs for cloud computing can be standardized. Comuzzi et al. [19] focus on contractual mechanisms of SLAs. They conducted a qualitative study interviewing industry experts to understand the extent to which SLA specifications in traditional environments can be applied to cloud computing. Muller et al. [20] present a design and implementation of SALMonADA, a service-based system to monitor and analyze SLAs to provide an explanation of violations. In SLA management and quality requirements representations, researchers do not use models at runtime thus making it difficult to monitor additional quality attributes when necessary or when SLAs change.

In conclusion, there is a lack of research that uses models at runtime with monitoring infrastructures to provide flexibility and independence to the monitoring process. Therefore, in this work, we present a monitoring infrastructure of cloud services that uses models at runtime to improve the fulfillment of SLAs.

3 Monitoring Process

The proposed monitoring process consists of three tasks, each of which is subdivided into particular activities. The process is based on the autonomic control loop technique. The idea of autonomic control loop is to measure system parameters, analyze them, plan corrective actions if necessary, and execute these actions in order to improve the system. One benefit of such an autonomic control loop is the reduced need for manual human intervention that often lead to low abstraction, maintenance, and reusability issues [11]. In this paper, we explain the process up until the *Analyze Results* task, which provides a report of SLA non-compliances, and in future research we will connect the monitoring middleware with a reconfiguration middleware in order to accomplish the autonomic control loop. The tasks which comprise our approach are presented in Fig. **1**; the monitoring process begins with the *Monitoring Configuration* task. The output of this task is the model at runtime which will be used for the monitoring middleware in the *Measurement Process* task.

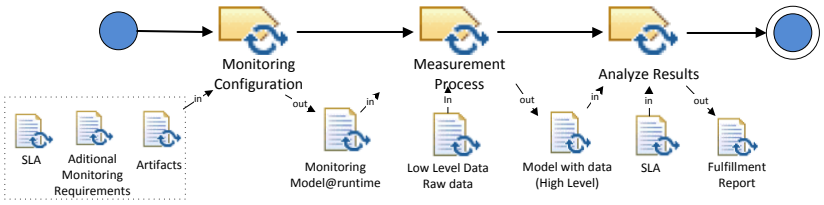

Fig. 1. Cloud Monitoring Process

The *Measurement Process* task captures low-level data from the running services using reflection techniques, and feeds the model at runtime with useful and filtered information, which is used by the *Analyze Results* task.

The *Analyze Results* task uses the data generated by the *Measurement Process*, compares it with the non-functional requirements specified in the SLA, and creates a *Fulfillment Report* that describes the non-compliances. The following sub-sections describe systematically each task and subtask of the monitoring process.

3.1 Monitoring Configuration

The Monitoring Configuration is responsible for the preparation of the model at runtime. It generates the code through a transformation. This code will be used by the monitoring middleware in order to operate with the data retrieved from the cloud.

Establish Monitoring Quality Requirements is the first task of this process. This task receives three artifacts as input: (1) the SLA with non-functional requirements, (2) additional monitoring requirements, and (3) the artifacts which will be analyzed by the monitoring process (e.g., services, applications). The output of this task is the *Monitoring Requirement Specification*. This contains characteristics and attributes that will be monitored. The *Quality Attributes Selection* uses as a guide a *SaaS Quality Model* to select the attributes specified in the *Monitoring Requirements Specification*. The *Measures Selection* task also uses a *SaaS Quality Model* and, depending on the user's perspective, selects the appropriate metrics to be applied. It is important to include the criticality related to the attributes, in order to take into account priority when taking corrective actions. Fig. 2. Monitoring Configurationshows the *Monitoring Configuration* task.

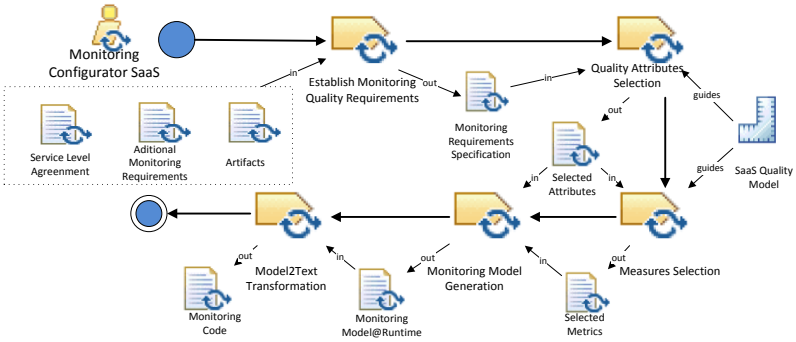

Fig. 2. Monitoring Configuration

The next step is the *Monitoring Model Generation*. The output of this task is a model at runtime that is the input of the *Model2Text Transformation task*. It generates the *Monitoring Code* with the model at runtime, which is used by the middleware in the *Monitoring Process.*

Fig. **3** shows the meta-model used by the *Monitoring Model Generation* task. Due to space constraints, we highlight only the most important meta-classes in the quality model at runtime:

- **RawReport:** contains the idCustomer, the idService, the monitoredExchangeId, the date and the timestamp of the data collected.

- **SaaSQualityModel:** contains the quality model reference, which provides all the attributes and metrics that can be applied in the monitoring middleware. Only a subset of the SaaSQualityModel attributes will be monitored.
- **MeasurableConcept:** can be a characteristic, sub-characteristic, or attribute that will be included in the monitoring process. Note that only the attributes can be measured and there is an OCL that specifies this constraint.
- **Metric:** is a measure of an attribute. A metric can be direct, indirect or an indicator and can have zero or many ways to be measured using *"operationalizations"*, depending on the attribute or the user perspective.
- **Operationalization:** is the way in which a metric is calculated. It can be a *MeasurementMethod*, a *CalculatingFunction*, a *Variable*, or a *Constant*.

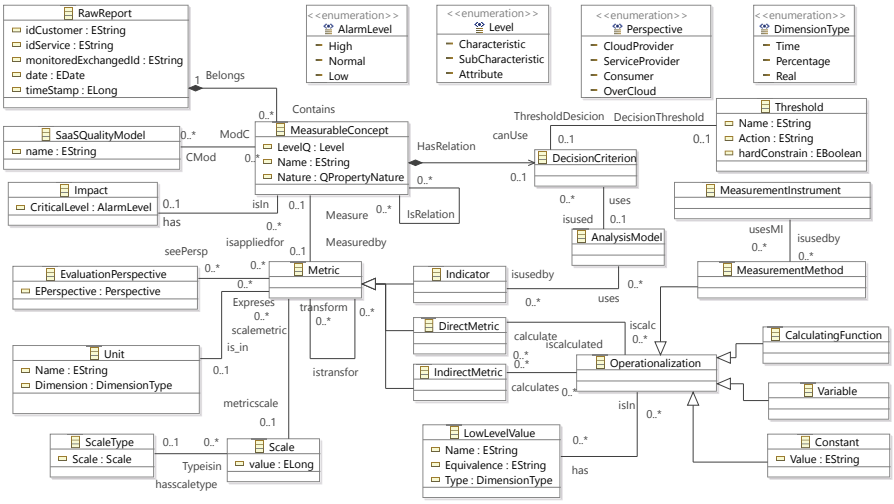

Fig. 3. Meta-model at runtime for the monitoring process

Lehmann et al. [21] argue that the meta-models of runtime must provide modeling constructs enabling the definition of: i) a prescriptive part of the model, specifying how the system should be: in this case, the prescriptive part can be related to the thresholds of the proposal meta-model; ii) a descriptive part of the model specifying how the system is. This is related to real values, which are retrieved from the services in the cloud in addition to the monitoring information contained in the *Raw Report*; iii) valid model modifications of the descriptive parts, executable at runtime. In this case, it may be necessary to retrieve new data about the state of the services, by adding new non-functional requirements to the monitoring process; iv) valid model modifications of the prescriptive parts, executable at runtime. This is the addition of new non-functional requirements and their thresholds in the model; v) causal connection: this is in the form of an information flow between the model and the services.

In order to achieve the descriptive and prescriptive model modifications, the development of a reconfiguration middleware is proposed as future research.

3.2 Measurement Process

The *Measurement Process* is included in a middleware that retrieves raw data from the services and applications and provides monitoring information to users and cloud providers. It uses the model at runtime defined in the previous section and uses a *Measurements Engine* to measure the attributes. The communication between services and the middleware is implemented using proxy elements or reflection techniques that allow the bidirectional communication between the monitoring infrastructure and the cloud services. The *Analysis Engine* receives information from the *Measurements Engine* and compares it with the SLA and non-functional requirements. The middleware provides results which can be used to take actions in order to improve the quality of the cloud and support SLA fulfillment. It is important to note that all of these processes represent overload to the cloud and should be correctly planned to avoid slowness. A middleware architecture enables communication and provides additional functionality such as improving control, monitoring and logging[11].

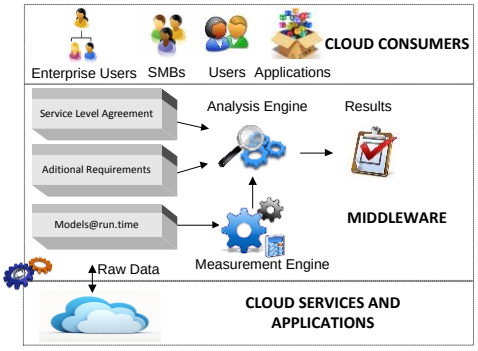

Fig. 4. Monitoring Architecture

3.3 Results Analysis

The *Analysis Engine* is part of the middleware and compares the values obtained by the monitoring process with the non-functional requirements, analyzing the results and reporting the analysis. Results obtained by the monitoring system may be used to plan a strategy to change the infrastructure using reconfiguration architectures that use, for example, an expert system or a knowledge base, adapting the system by itself and supporting the fulfillment of non-functional requirements, closing the autonomic control loop. However this is reserved for future research.

4 Example

In this section, the monitoring process is illustrated through an example that implements all the steps involved in our strategy.

The monitoring process can be applied to any cloud platform. For this example, Azure platform is used [22]. This is a services platform hosted by Microsoft data centers, which provides a platform as a service and a set of developer services; Azure

also enables the building, deploying and managing of services, which can be developed in any language, tool or framework and integrate public cloud applications using existing IT environments. Moreover, Azure has a library called Diagnostics that allows retrieval of diagnostic data from the cloud infrastructure.

This example uses Azure to provide an online auction site with services. In these kinds of applications users demand characteristics very related with cloud environments, and it is necessary to monitor them; for example availability, and another characteristics such as scalability and elasticity, which are very important and specific for cloud scenarios. The availability requirement will be focused on in this auction site.

4.1 Monitoring Configuration

Establish Monitoring Quality Requirements is the first task in the *Monitoring Configuration*. For this example, we consider that the SLA includes availability as a non-functional requirement. The server provider commits that the bid service will be available 99.50% or more of the time in a given calendar month. If the service offered fails to meet this commitment, the server provider will apply a service credit to the customer account. Additional monitoring requirements will be not considered, and the artifact to be monitored is the bid service. For both, the *Quality Attributes Selection* and *Measures Selection*, we can use quality models specific for cloud computing services [10] or third part studies that define attributes and metrics for specific attributes [23]. The availability is studied in [10] and this attribute is measured by the Robustness of Service (ROS) metric. The ROS metric is computed by [10] as (1):

$$ROS = \frac{(available\ for\ invoking\ SaaS)}{(total\ time\ for\ operating\ SaaS)} \quad (1)$$

The range is 0...1 and the higher value, the higher availability the SaaS has [10].

Once this information is obtained, the *Monitoring Model Generation* and *Model to Text Generation* tasks are performed in order to generate the model at runtime for the *Measurement Process*. In our example, the availability is categorized as critical because in the auction domain, availability is essential as it represents money.

4.2 Measurement Process

The *Measurement Process* is the central part of the middleware and uses the model at runtime generated by the previous step. This process calculates the ROS value, which is the metric selected in the previous task, taking into account the values collected from the bid service by the Diagnostics Tool in the Azure Platform, which is an implementation of the proxy mechanism described in Section 3.2. It is possible to apply the proposed process to any attribute by selecting the appropriate metric. Sometimes it may be necessary to use past information, in metrics that use intervals, it is possible to access the past instances of the model at runtime, (e.g. measuring the scalability).

4.3 Results Analysis

The Analysis Engine compares the non-functional requirements specified in the SLA

with real values resulting from the *Measurement Process*. For this example, the service provider offers in the bid service 99.5% of availability and so by comparing the result with the SLA, we can conclude if the service fulfills the agreement. If the availability of the service described in the SLA is fulfilled, a periodical or on demand report can be generated. However, if the availability requirement is not fulfilled, the monitoring middleware sends an alarm signal. A report with non-compliances is generated, detailing alarms being triggered and the criticality of the monitored attribute.

5 Conclusions and Future Work

In this paper, we have introduced a monitoring process using models at runtime, in which it is possible to specify non-functional requirements described by a SLA, as well as other non-functional requirements of interest to server providers. We have described the meta-model of the model at runtime which will be used in the process and have discussed the important parts of the model at runtime which are integrated into the monitoring process. This approach is useful in measuring higher-level attributes specified by SLAs, and it provides flexibility when the evaluator needs to change or add non-functional requirements since changes will be done in the model at runtime and the monitoring infrastructure will not need to be affected.

As future work, we plan to implement this middleware, defining all input and output artifacts involved in the process (e.g., SLAs, models at runtime, etc.) and investigate in practice how the models at runtime will behave when non-functional requirements are modified. Finally, our objective is to provide guidelines to support the definition of the model at runtime from SLAs and to determine what actions can be performed when violations of the SLA clauses arises. Through this line of research, we will explore what dynamic architecture reconfigurations are possible in order to improve the overall quality of the cloud application, and in this way, to complete the autonomic control loop for the self-adaptation of high-quality services in the cloud.

6 Acknowledgments

This research is supported by the Value@Cloud project (TIN2013-46300-R), the ValI+D program (ACIF/2011/235) from the Generalitat Valenciana; the Scholarship Program Senescyt-Ecuador; and University of Cuenca, Ecuador.

References

1. Information Technology Infrastructure Library (ITIL), http://www.itil-officialsite.com
2. Correia, A., e Abreu, F.: Model-Driven Service Level Management. Mechanisms for Autonomous Management of Networks and Services. pp.85–88. Berlin Heidelberg (2010)
3. Keller, A., Ludwig, H.: The WSLA Framework: Specifying and Monitoring Service Level Agreements for Web Services. J. Netw. Syst. Manag. 11, 57–81 (2003)
4. Emeakaroha, V.C., Brandic, I., Maurer, M., Dustdar, S.: Low level Metrics to High level SLAs - LoM2HiS framework: Bridging the gap between monitored metrics and SLA

parameters in cloud environments. Int. Conf. on High Performance Computing and Simulation (HPCS), pp. 48–54. Caen, France (2010)
5. Baresi, L., Ghezzi, C.: The Disappearing Boundary Between Development-time and Run-time. Workshop on Future of Software Engineering Research FSE/SDP. pp. 17–22. ACM, Santa Fe, New Mexico, USA (2010)
6. Cheng, B.C., Eder, K., Gogolla, M., Grunske, L., Litoiu, M., Müller, H., Pelliccione, P., Perini, A., Qureshi, N., Rumpe, B., Schneider, D., Trollmann, F., Villegas, N.: Using Models at Runtime to Address Assurance for Self-Adaptive Systems, http://dx.doi.org/10.1007/978-3-319-08915-7_4
7. Gao, J., Pattabhiraman, P., Bai, X., Tsai, W.T.: SaaS performance and scalability evaluation in clouds. 6th Int. Symposium on Service Oriented System Engineering (SOSE), pp. 61–71. Irvine, CA, USA (2011)
8. Bencomo, N., Blair, G., Götz, S., Morin, B., Rumpe, B.: Report on the 7th Int. Workshop on Models@Runtime. SIGSOFT Softw. Eng. Notes. 38, 27–30. Innsbruck, Austria (2013).
9. Blair, G., Bencomo, N., France, R.B.: Models@ run.time. Computer (Long. Beach. Calif). 42, 22–27 (2009)
10. Lee, J.Y., Lee, J.W., Cheun, D.W., Kim, S.D.: A Quality Model for Evaluating Software-as-a-Service in Cloud Computing. 7th ACIS Int. Conf. on Software Engineering Research, Management and Applications. pp. 261–266, Haikou, China (2009)
11. Szvetits, M., Zdun, U.: Systematic literature review of the objectives, techniques, kinds, and architectures of models at runtime. Softw. Syst. Model. 1–39 (2013)
12. Bencomo, N., France, R.B., Götz, S., Rumpe, B.: Summary of the 8th International Workshop on Models @ Run.time. MoDELS@Runtime. , Miami, FL, USA (2013)
13. Bertolino, A., Calabrò, A., Lonetti, F., Di Marco, A., Sabetta, A.: Towards a Model-Driven Infrastructure for Runtime Monitoring. In: Troubitsyna, E. (ed.) Software Engineering for Resilient Systems. pp. 130–144. Springer Berlin Heidelberg (2011)
14. Bertolino, A., Calabrò, A., Lonetti, F., Sabetta, A.: GLIMPSE: A Generic and Flexible Monitoring Infrastructure. 13th European Workshop on Dependable Computing. pp. 73–78, Pisa, Italy (2011)
15. Bencomo, N., Whittle, J., Sawyer, P., Finkelstein, A., Letier, E.: Requirements reflection: requirements as runtime entities. 32nd Int. Conf. on Soft. Eng. pp. 199–202, Cape Town, South Africa (2010)
16. 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. 36th Annual Computer Software and Applications Conference. pp. 499–508, Izmir, Turkey (2012)
17. Correia, A., e Abreu, F.B., Amaral, V.: SLALOM: a Language for SLA specification and monitoring. CoRR. abs/1109.6, (2011)
18. Myerson, J.: Best practices to develop SLAs for cloud computing, http://ibm.com/developerWorks/
19. Comuzzi, M., Jacobs, G., Grefen, P.: Clearing the Sky - Understanding SLA Elements in Cloud Computing, Eindhoven, Nederland (2013)
20. Muller, C., Oriol, M., Franch, X., Marco, J., Resinas, M., Ruiz-Cortes, A., Rodriguez, M.: Comprehensive Explanation of SLA Violations at Runtime. Serv. Comput. IEEE Trans. 7, 168–183 (2014)
21. Lehmann, G., Blumendorf, M., Trollmann, F., Albayrak, S.: Meta-modeling Runtime Models. Int. Conf. on Models in Software Engineering. pp. 209–223. Oslo, Norway (2010)
22. What Is Azure?, https://azure.microsoft.com/en-us/overview/what-is-azure/.
23. Xiong, K., Perros, H.: Service Performance and Analysis in Cloud Computing. IEEE Congress on Services. pp. 693–700, LA, California, USA (2009)