

# A Model-Based Framework for SLA Management and Dynamic Reconfiguration

Mahin Abbasipour<sup>1</sup>(✉), Ferhat Khendek<sup>1</sup>, and Maria Toeroe<sup>2</sup>

<sup>1</sup> ECE, Concordia University, Montréal, Canada  
{mah\_abb,khendek}@encs.concordia.ca

<sup>2</sup> Ericsson Inc., Montréal, Canada  
maria.toeroe@ericsson.com

**Abstract.** A Service Level Agreement (SLA) is a contract between a service provider and a customer that defines the expected quality of the provided services, the responsibilities of each party, and the penalties in case of violations. In the cloud environment where elasticity is an inherent characteristic, a service provider can cater for workload changes and adapt its service provisioning capacity dynamically. Using this feature one may provide only as many resources as required to satisfy the current workload and SLAs, the system can shrink and expand as the workload changes. In this paper, we introduce a model-based SLA monitoring framework, which aims at avoiding SLA violations from the service provider side while using only the necessary resources. We use UML models to describe all the artifacts in the monitoring framework. The UML models not only increase the level of abstraction but they are also reused from the system design/generation phase. For this purpose, we develop metamodels for SLAs and for monitoring. In the monitoring framework, all abstract SLA models are transformed into an SLA compliance model which is used for checking the compliance to SLAs. To avoid SLA violations as well as resource wasting, dynamic reconfigurations are triggered as appropriate based on the predefined Object Constraint Language (OCL) constraints using thresholds.

**Keywords:** Monitoring · Elasticity · SLA violation avoidance · Model driven engineering · OCL constraints

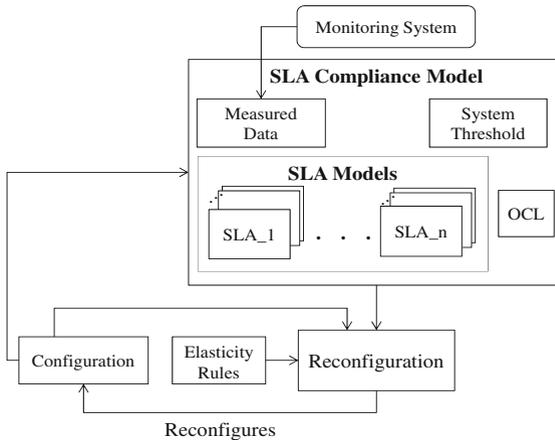
## 1 Introduction

A Service Level Agreement (SLA) is a contract between a customer and a service provider that aims at describing the level of service quality and the obligations of each party in the agreement. An SLA violation occurs when any of the parties fails to meet their obligations [14]. A violation may be associated with a penalty.

During the operation of a system its workload changes dynamically, which results in variable resource usage. To increase revenue, instead of allocating a fixed amount of resources, providers try to allocate only as much as required to satisfy the current customer needs and adapt subsequently to the workload

changes. This could be challenging as one would like to provision resources not too early or too much to avoid the waste of resources, and not too late or inadequately to avoid SLA violations. To accurately adapt the system at runtime, the system should be monitored: The metrics or events of interest are collected and after evaluation actions are triggered to modify the managed system accordingly.

Figure 1 gives an overall view of our SLA management framework. In this framework, the system is scaled according to the workload variations while avoiding SLA violations. For this, the system configuration and the related elasticity rules are generated offline during the design phase and determine the configuration changes needed to scale up/down (adding/removing resources of a node) and in/out (removing/adding a node). In this framework, all the SLAs, their corresponding measurements and the thresholds are combined into an SLA compliance model. The validation of the SLA compliance model may generate triggers for scaling down and/or in the system to save resources when the workload decreases or for increasing the resources when workload goes up to avoid SLA violations. The monitoring system feeds the framework with the measured data. The thresholds are related to the maximum and the minimum capacity of the current system configuration and are used to check if the system needs reconfiguration. Accordingly, the values for the thresholds are re-evaluated with each reconfiguration of the system. To map the measured data to SLA parameters and to generate the triggers for the reconfiguration OCL [10] constraints have been defined. When a trigger is generated the appropriate elasticity rules are invoked to reconfigure the system with the minimum required resources. In this short paper we target primarily, but not limited to, component based systems deployed on a number of virtual machines typical for cloud computing. Furthermore, we focus on the modeling aspects and the use of OCL constraints to trigger dynamic reconfiguration.



**Fig. 1.** SLA compliance management and dynamic reconfiguration

In this short paper, we introduce the principles of our approach for SLA compliance monitoring and dynamic reconfiguration. The rest of the paper is organized as follows. In Sect. 2, the metamodels for SLA and SLA compliance are presented. Section 3 explains how OCL constraints are used to generate dynamic reconfiguration triggers. We discuss the related work in Sect. 4 and conclude in Sect. 5.

## 2 Modeling for SLA Compliance Management

To manage the compliance to SLAs, the running system needs to be monitored, data has to be collected and the SLAs checked periodically. We adopt a model driven approach not only to facilitate the understanding, design and maintenance of the system [9], but also to reuse the models generated during the system design phase such as the system configuration, and to build on existing tools. We define our metamodels using Papyrus [5]. The Atlas Transformation Language (ATL) [6] is used to combine all SLA models into an SLA compliance model. In this section, we introduce the metamodels for SLA and for SLA compliance.

### 2.1 SLA Metamodel

The SLA metamodel is shown in Fig. 2. Each SLA has an ID and is an agreement between a provider and a customer. A third party may also participate to verify the agreed Service Level Objectives (SLO) and play the monitoring role [7]. An SLA is applicable for a specific time duration and has a cost. This cost can be a constant value or it can be a function based on the usage of the services. An SLA includes some service functionalities that the provider agrees to provide with specific Quality of Service (QoS). An abstract stereotype *SlaParameter* captures the different types of QoS the customer and the provider agree on. The agreed values are represented by *maxAgreedValue* and *minAgreedValue* in the figure. For example, the *maxAgreedValue* in the SLA parameter *DataRate* represents the maximum number of requests per second the customer may send for the specific service.

The monitoring system measures each metric (*MeasuredMetric*) at a predefined frequency. Customers may also want to specify how often the SLA parameters are monitored and checked. This customization is represented by *SlaMetric* stereotype. However, it should be compatible with the capability of the monitoring system. In other words, the frequency agreed on in the SLA must be less or equal to the frequency of measurements of the monitoring system.

### 2.2 SLA Compliance Metamodel

An SLA compliance model is the combination of all SLA models, part of the configuration model and the measurements obtained from the monitoring system. The main reason for merging all SLA models into one model is that we

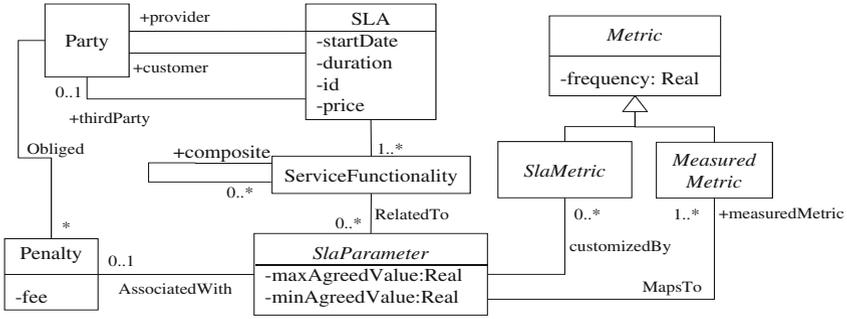


Fig. 2. SLA metamodel

want to be able not only to avoid violations in each individual SLA but also to trigger elasticity rules which are related to all customers resource usage.

The SLA compliance metamodel is shown in Fig. 3. The same service with the same or different SLA parameters is generally offered to multiple customers. The *MeasuredMetric* stereotype represents the value the monitoring system measures per service for each customer or per node of the system. When an SLA parameter related to a service is not respected, the *BelongsTo* relation indicates which SLA has been violated.

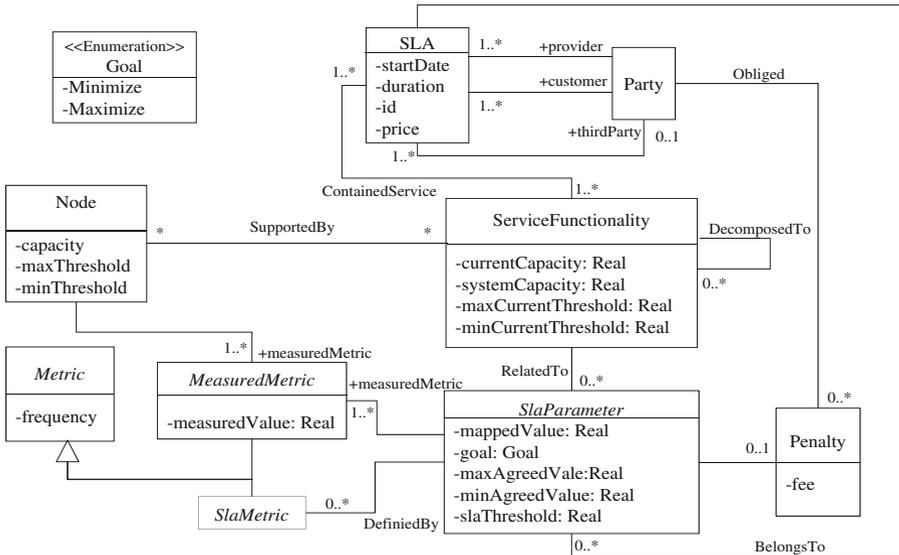


Fig. 3. SLA compliance metamodel

The monitoring system collects raw metrics. Some of these metrics (e.g. service up/down time) and the SLA parameters perceived by the customers (e.g. availability of service) are not at the same level. To bridge the gap between the measured values and SLA parameters, OCL constraints have been defined as mapping rules. The attribute *mappedValue* represents the value of such mapped measurements. A service may be a composition of different service functionalities, which may be mapped similarly or measured at the composite level.

The attribute *goal* of an SLA parameter specifies the parameters optimization goal. For some SLA parameters, like availability, the optimization goal is maximization while for others like response time, the goal is minimization. We categorize our OCL constraints for SLA violation avoidance based on these optimization goals. When a new SLA parameter is introduced and taken into consideration, there is no need for new OCL constraints as long as its optimization goal fits into one of the aforementioned categories.

### 3 Dynamic Reconfiguration

In the proposed framework, OCL constraints are used to trigger dynamic reconfiguration. The OCL constraints are defined on a number of attributes: The attribute *currentCapacity* in the *ServiceFunctionality* stereotype specifies the maximum workload (e.g. requests/second) the system in its current configuration can handle for a specific service. The attribute *systemCapacity* is defined at the design phase as the maximum system capacity for the service. This is the maximum capacity the system can be expanded to without major changes (e.g. upgrade/re-design). As mentioned earlier to avoid SLA violations and trigger reconfiguration, we use thresholds. Some of the thresholds are related to all customers (aggregate) resource usage while others are related to individual SLAs.

- *maxCurrentThreshold* and *minCurrentThreshold*: For each service, the system is dimensioned dynamically with a *currentCapacity* to handle a certain workload. In order to avoid SLA violations, i.e. workload exceeding *currentCapacity*, we define a *maxCurrentThreshold* point (with *maxCurrentThreshold* < *currentCapacity*) at which the system capacity is increased by scaling it up/out to a new *currentCapacity* and for which a new *maxCurrentThreshold* is defined. Therefore, the relation *workload* < *maxCurrentThreshold* must be respected. Not to waste resources we also define a *minCurrentThreshold* where we scale down/in the capacity of the system to a lower capacity (i.e. the relation *workload* > *minCurrentThreshold* must be respected). We use OCL constraints to define these restrictions. As a result the violation of the defined OCL constraints triggers the scaling of the system. In this paper, we assume that the service workload is distributed evenly in the system.
- *slaThreshold*: Some SLA parameters like service availability are set on a per customer basis. Therefore, to avoid SLA violations, we need to watch the SLAs separately using a *slaThreshold* for each SLA. These parameters behave similarly with respect to violation. Some of them like availability and throughput

for which a higher value is preferable (i.e. the attribute goal is equal to Maximize) will be violated by a service provider when in the SLA compliance model, the experienced quality is less than the *slaThreshold* (i.e. the relation *mappedValue* > *slaThreshold* must be respected all the time if *goal=Maximize*); while for others like response time, the violation happens from the service provider side when the measured response time is greater than the *slaThreshold* (i.e. the relation *mappedValue* < *slaThreshold* must be respected if *goal=Minimize*). Again, we use OCL constraints to define these restrictions. By violation of these OCL constraints, triggers will be generated to avoid SLA violations.

The following thresholds are related to the node resource utilization:

- *maxThreshold* and *minThreshold*: To avoid SLA violations because of node limitations, e.g. the load on a node exceeding its capacity, we define the *maxThreshold* point at which we allocate more resources to the node (e.g. virtual machine, hyper scale system) or add more nodes to the system (i.e. the relation *load* < *maxThreshold* should be respected). To avoid the wasting of resources, the *minThreshold* is used to reduce the node resources, for example, by removing a node or decreasing the virtual resources of a virtual machine. The addition/removal of resources to/from the node increases/reduces the capacity of the node and therefore new thresholds are defined. The *maxThreshold* and *minThreshold* are vectors where different types of node resources (e.g. CPU, RAM, etc.) are taken into account.

To obtain the SLA compliance model from the individual SLA models, we use model transformation. As the number of SLA models varies over time, SLAs may be added or removed, we use different model transformations. For the addition, the initial SLA compliance model is obtained directly from the first SLA model by transformation that creates all the model elements. Subsequent SLA models are added incrementally using another transformation which takes into account the already existing elements of the SLA compliance model. Similarly, when an SLA is removed, the elements related to only this SLA should be removed from the SLA compliance model together with their measurements. This is achieved with a different transformation that takes the SLA to be removed and the SLA compliance model as input and generates a new SLA compliance model. In the current prototype implementation the addition and removal of SLAs are done offline.

## 4 Related Work

There are a number of works that define languages for modeling SLAs. Most of the languages are for a specific domain. For example in [14, 15], the authors define SLAng suitable for network services. Others like [7, 12] focus on web services. QML [4] allows customers to define the SLA parameters to be measured for monitoring purposes. Since a customer defined parameter may not be observable for a specific system and an agreement needs a common understanding of

parameters between parties, this can result in inconsistency with the monitoring systems capability. In our proposal, not only do we allow customers to customize their SLAs but we also make sure that this customization is compatible with the capabilities of the monitoring system.

In [2], a metamodel for SLA description and monitoring is proposed but it is not clear how the compliance to SLAs is checked. In [11], a timed automata is used to detect violations with respect to response time. The work in [14] is closely related to this paper. In [14], to detect SLA violations, different OCL constraints for different SLA parameters have been defined. However, to add a new parameter to an SLA a new OCL constraint for the violation detection has to be added as well, which is not the case in our framework. In [14], SLA compliance is the only goal, while in our case we want to achieve this goal with the minimum amount of resources needed for the workload at any given time and to grow/shrink the system according to the workload variations.

Monitoring and scaling of cloud systems based on the demand has been extensively investigated. However, only a few works have looked into SLA compliance at an abstract level. In [3], a framework for monitoring SLAs is proposed. It consists of three components: a monitoring system for providing measurements, LoM2HiS for mapping monitored data to parameters in the SLAs, and a knowledge database which uses past experience to solve current SLA related issues. This framework is suitable for the infrastructure layer of the cloud. Similarly, [1] focuses on the infrastructure level only but nothing is done at the other layers to respond to application level workload variations. In our framework both, infrastructure and application, levels are handled. On the other hand, [8, 13] for instance do not take SLAs into account.

## 5 Conclusion and Future Work

Service providers aim at increasing their revenue by operating a system with the minimum amount of resources necessary to avoid SLA violation penalties. For this purpose, there is a need for an SLA management and dynamic reconfiguration framework that scales the system (up/down and in/out) according to the workload changes while avoiding SLA violations. In this paper, we proposed such a framework. It is model driven, it is at the right level of abstraction. OCL constraints are written for categories of parameters and are not specific for each parameter, which eases future extension. More important, the proposed framework reuses models developed at the system design stage. This work is at an early stage, more investigations are required, for instance, to generate the elasticity rules automatically and to handle the correlation of the generated triggers. We also need to investigate the challenging issue of SLA compliance model evolution at run time, i.e. addition and removal of SLA models while the system is in operation. The performance of such a model based framework needs to be assessed as well.

**Acknowledgments.** This work has been partially supported by Natural Sciences and Engineering Research Council of Canada (NSERC) and Ericsson.

## References

1. Ali-Eldin, A., Tordsson, J., Elmroth, E.: An adaptive hybrid elasticity controller for cloud infrastructures. In: Network Operations and Management Symposium (NOMS), pp. 204–212. IEEE (2012)
2. Debusmann, M., Kroger, R., Geihs, K.: Unifying service level management using an MDA-based approach. In: Network Operations and Management Symposium (NOMS), IEEE/IFIP, Vol. 1, pp. 801–814. IEEE (2004)
3. Emeakaroha, V.C., et al.: Towards autonomic detection of SLA violations in cloud infrastructures. *Future Gener. Comput. Syst.* **28**(7), 1017–1029 (2012). Elsevier
4. Frolund, S., Koistinen, J.: Qml: a language for quality of service specification. Hewlett-Packard Laboratories (1998)
5. Gérard, S., Dumoulin, C., Tessier, P., Selic, B.: 19 Papyrus: a UML2 tool for domain-specific language modeling. In: Giese, H., Karsai, G., Lee, E., Rumpe, B., Schätz, B. (eds.) *Model-Based Engineering of Embedded Real-Time Systems*. LNCS, vol. 6100, pp. 361–368. Springer, Heidelberg (2010)
6. Jouault, F., Kurtev, I.: Transforming models with ATL. In: Bruel, J.-M. (ed.) *MoDELS 2005*. LNCS, vol. 3844, pp. 128–138. Springer, Heidelberg (2006)
7. Keller, A., Ludwig, H.: The WSLA framework: specifying and monitoring service level agreements for web services. *J. Netw. Syst. Manag.* **11**(1), 57–81 (2003). Springer
8. König, B., Calero, J.A., Kirschnick, J.: Elastic monitoring framework for cloud infrastructures. *IET Commun.* **6**(10), 1306–1315 (2012)
9. MDA User Guide, version 1.0.0, OMG (2003)
10. OMG Object Constraint Language (OCL), version 2.3.1, OMG, January 2012
11. Raimondi, F. et al.: A Methodology for on-line monitoring non-functional specifications of web-services. In: *First International Workshop on Property Verification for Software Components and Services (PROVECS)*, pp. 50–59 (2007)
12. Sahai, A., Machiraju, V., Sayal, M., van Moorsel, A., Casati, F.: Automated SLA monitoring for web services. In: Feridun, M., Kropf, P.G., Babin, G. (eds.) *DSOM 2002*. LNCS, vol. 2506, pp. 28–41. Springer, Heidelberg (2002)
13. Sedaghat, M., Hernandez-Rodriguez, F., Elmroth, E.: A virtual machine re-packing approach to the horizontal vs. vertical elasticity trade-off for cloud autoscaling. In: *2013 ACM Cloud and Autonomic Computing Conference* (2013)
14. Skene, J., Emmerich, W.: Generating a contract checker for an SLA language (2004). [https://www.researchgate.net/publication/32885283\\_Generating\\_a\\_contract\\_checker\\_for\\_an\\_SLA\\_language](https://www.researchgate.net/publication/32885283_Generating_a_contract_checker_for_an_SLA_language)
15. Skene, J., Lamanna, D.D., Emmerich, W.: Precise service level agreements. In: *26th International Conference on Software Engineering*, pp. 179–188. IEEE Computer Society (2004)



<http://www.springer.com/978-3-319-24911-7>

SDL 2015: Model-Driven Engineering for Smart Cities  
17th International SDL Forum, Berlin, Germany, October  
12-14, 2015, Proceedings

Fischer, J.; Scheidgen, M.; Schieferdecker, I.; Reed, R.  
(Eds.)

2015, XIV, 285 p. 90 illus. in color., Softcover

ISBN: 978-3-319-24911-7