Low Level Metrics to High Level SLAs - LoM2HiS Framework: Bridging the Gap Between Monitored Metrics and SLA Parameters in Cloud Environments

Vincent C. Emeakaroha, Ivona Brandic, Michael Maurer, Schahram Dustdar
Information Systems Institute, Vienna University of Technology
{vincent, ivona, maurer, dustdar}@infosys.tuwien.ac.at

ABSTRACT

Cloud computing represents a novel on-demand computing approach where resources are provided in compliance to a set of predefined non-functional properties specified and negotiated by means of Service Level Agreements (SLAs). In order to avoid costly SLA violations and to timely react to failures and environmental changes, advanced SLA enactment strategies are necessary, which include appropriate resource-monitoring concepts. Currently, Cloud providers tend to adopt existing monitoring tools, as for example those from Grid environments. However, those tools are usually restricted to locality and homogeneity of monitored objects, are not scalable, and do not support mapping of low-level resource metrics e.g., system up and down time to high-level application specific SLA parameters e.g., system availability. In this paper we present a novel framework for managing the mappings of the Low-level resource Metrics to High-level SLAs (LoM2HiS framework). The LoM2HiS framework is embedded into FoSII infrastructure, which facilitates autonomic SLA management and enforcement. Thus, the LoM2HiS framework detects future SLA violation threats and can notify the enactor component to act so as to avert the threats. We discuss the conceptual model of the LoM2HiS framework, followed by the implementation details. Finally, we present the first experimental results and a proof of concept of the LoM2HiS framework.

KEYWORDS: Cloud Computing, Resource monitoring, SLA mapping, SLA enforcement, SLA management, Performance evaluation.

1. INTRODUCTION

In recent years, Cloud computing has become a key IT megatrend that will take root, although it is in infancy in terms of market adoption. Cloud computing can be defined as the convergence and evolution of several concepts from virtualization, distributed application design, Grid, and enterprise IT management to enable a more flexible approach for deploying and scaling applications [3].

Cloud services include high performance applications requiring lots of system resources. Service provisioning in the Cloud is based on Service Level Agreements (SLA), which is a contract signed between the customer and the service provider. It states the terms of the service including the non-functional requirements of the service specified as quality of service (QoS), obligations, service pricing, and penalties in case of agreement violations.

In order to guarantee an agreed SLA, the service provider must be capable of monitoring its infrastructure (host) resource metrics to enforce the agreed service terms. Traditional monitoring technologies for single machines or Clusters are restricted to locality and homogeneity of monitored objects and, therefore, cannot be applied in the Cloud in an appropriate manner. Moreover, in traditional systems there is a gap between monitored metrics, which are usually low-level entities, and SLA agreements, which are high-level user guarantee parameters.

In this paper we present a novel framework for the mapping of Low-level resource Metric to High-level SLA parameters named *LoM2HiS* framework. *LoM2HiS* framework is embedded into *FoSII* infrastructure aiming at developing an infrastructure for autonomic SLA management and enforcement. Thus, *LoM2HiS* represents the first building block of the *FoSII* [4] infrastructure. We present the conceptual design of the framework including the run-time and host monitors, the enactor component, and the SLA mapping database. We discuss our novel communication model based on queuing networks ensuring the scalability of the *LoM2HiS* framework. Moreover, we demonstrate sample mappings from the low-level resource monitoring metrics to the SLA parameters. Thereafter, we discuss the implementation issues and finally, we present first experimental results as proof of concept.

The rest of the paper is organized as follows. Section 2 presents the related work. In Section 3 we present the conceptual model of the framework. In Section 4 implementation issues of the framework are described. It also describes the communication model within the framework. Section 5 deals with the framework evaluation based on a real testbed and the discussion of the achieved results. Section 6 presents the conclusion of the paper and our planned future research work.

2. RELATED WORK

We classify related work on enforcement of Cloud based services into (i) monitoring of Cloud/Grid/Web services [12,15,1], (ii) SLA management including QoS management [8-11] and (iii) mapping techniques of monitored metrics to SLA parameters and attributes [20,13,14]. Since there is very little work on monitoring, SLA management, and metrics mapping in Cloud systems we look particularly into related areas such as Grid and SOA based systems.

Dobson et al. [12] present a unified quality of service (QoS) ontology applicable to the main scenarios identified such as QoS-based Web services selection, QoS monitoring and QoS adaptation. Comuzzi et al. 2009 [15] define the process for SLA establishment adopted within the EU project *SLA@SOI* framework. The authors propose the architecture for monitoring of SLAs considering two requirements introduced by SLA establishment: the availability of historical data for evaluating SLA offers and the assessment of the capability to monitor the terms in an SLA offer. NetLogger [1] is a distributed monitoring system, which monitors and collects information from networks. Applications can invoke NetLogger's API to survey the overload before and after some request or operation. However, it monitors only network resources.

Theilman et al. 2008 [9] discuss an approach for multi-level SLA management, where SLAs are consistently specified and managed within a service-oriented infrastructure (SOI). They present the run-time functional view of the conceptual architecture and discuss different case studies including Enterprise Resource Planning (ERP) or financial services. Koller et al. 2009 [8] discuss autonomous QoS management using a proxy-like approach. The implementation is based on WS-Agreement. Thereby, SLAs can be exploited to define certain QoS parameters that a service has to maintain during its interaction with a specific customer. However, their approach is limited to Web services and does not consider requirements of Cloud Computing infrastructures like scalability. Frutos et al. 2009 [10] discuss the main approach of the EU project *BREIN* [11] to develop a framework, which extends the characteristics of computational Grids by driving their usage inside new target areas in the business domain for advanced SLA management. However, BREIN applies SLA management to Grids, whereas we target SLA management in Clouds.

Brandic et al. 2009 [20] present an approach for adaptive generation of SLA templates. Thereby, SLA users can define mappings from their local SLA templates to the remote templates in order to facilitate communication with numerous Cloud service providers. However, they do not investigate mapping of monitored metrics to agreed SLAs. Rosenberg et al. 2006 [13] deal with QoS attributes for Web services. They identified important QoS attributes and their composition from resource metrics. They presented some mapping techniques for composing QoS attributes from resource metrics to form SLA parameters for a specific domain. However, they did not deal with monitoring of resource metrics. Bocciarelli et al. 2007 [14] introduce a model-driven approach for integrating performance prediction into service composition processes carried out using BPEL. In their approach, they composed service SLA parameters from resource metrics using some mapping techniques. But they did neither consider resource metrics – nor SLA monitoring.

To the best of our knowledge, none of the discussed approaches deal with mappings of low-level monitored metrics to high-level SLA guarantees as those necessary in Cloud-like environments.

3. DESIGN OF THE LOM2HIS FRAMEWORK

The LoM2HiS framework is the first step towards achieving the goals of the *FoSII* infrastructure. In the sections below, we present an overview of the *FoSII* infrastructure and give details of the *LoM2HiS* framework design.

3.1. FoSII Infrastructure Overview

Foundations of Self-governing ICT Infrastructures (*FoSII)* is an ongoing research project at Vienna University of Technology [4]. It proposes models and concepts for autonomic SLA management and enforcement. Figure 1 depicts the *FoSII* infrastructure. It is used to manage the whole lifecycle of self-adaptable Cloud services [17]. Each *FoSII* service implements three interfaces: (i) negotiation interface necessary for the establishment of SLA agreements, (ii) job-management interface necessary to start the job, upload data, and similar job management actions, and (iii) the self-management interface necessary to devise actions in order to prevent SLA violations. It specifies operations for sensing changes of the desired states using the host monitor and run-time monitor sensors (arrow *a* and *b* in Figure 1) and for reacting to those changes.

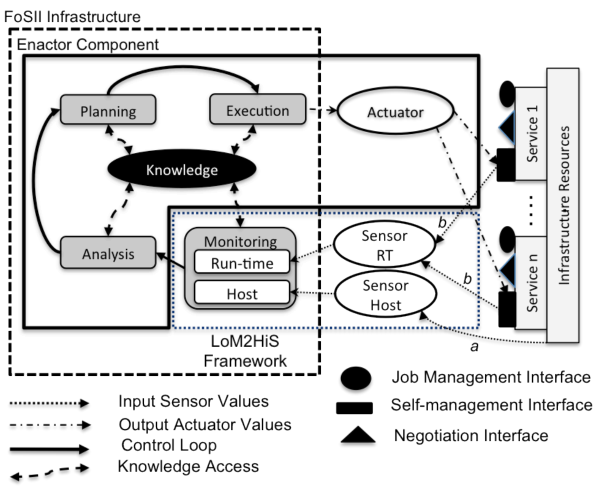

Figure 1. FoSII Infrastructure Overview

Logically, *FoSII* infrastructure consists of two core parts: i) the *Enactor Component*, which represents the self-management component for the deployed services, and ii) the *LoM2HiS framework*, which provides monitoring information for the enactor component.

3.2. Enactor Component

As shown in Figure 1 this component is the autonomic part of *FoSII* infrastructure. It is based on the principles of autonomic computing. In autonomic systems, humans do not control the system. Moreover, they define the general policies and rules that serve as input for the self-management process. Such systems constantly adapt themselves to changing environmental conditions like workload, hardware, and software failures [18]. An important characteristic of an autonomic system is an intelligent closed loop of control. As depicted in Figure 1, the autonomic manager manages the elements' states and behaviours. Typically, control loops are implemented following MAPE (Monitoring, Analysis, Planning, and Execution) steps [17]. The human defined policies and rules with which the MAPE processes are guided, are placed in knowledge databases (component "knowledge" in Figure 1). These rules evolve and adapt to environmental changes. The goal of this component is to achieve an autonomic SLA management, where appropriate actions are taken to prevent future SLA violations.

3.3. Overview LoM2HiS Framework

In this framework, we assumed that the SLA negotiation process is completed and the agreed SLAs are stored in the repository for service provisioning. Beside the SLAs, the predefined threat thresholds are also stored in a repository. The concept of detecting future SLA violation threats is designed by defining more restrictive thresholds known as threat thresholds that are stricter than the normal SLA objective violation thresholds. Generation of the threat thresholds is far from trivial and is part of our ongoing work. In this paper we assume predefined threat thresholds.

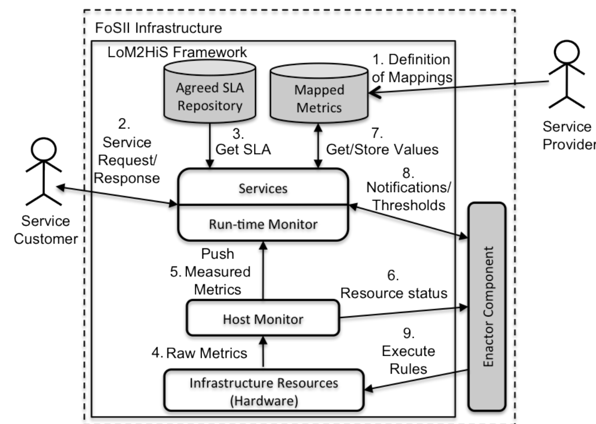

Figure 2. LoM2HiS Framework Architecture

Figure 2 presents the architecture of our *LoM2HiS* framework. The service component including the run-time monitor represents the application layer where services are deployed using a Web Service container e.g., Apache Axis. The run-time monitor is designed to monitor the services based on the negotiated and agreed SLAs. After agreeing on SLA terms, the service provider creates mapping rules for the *LoM2HiS* mappings (step 1 in Figure 2) using Domain Specific Languages (DSLs). DSLs are small languages that can be tailored to a specific problem domain. Once the customer requests the provisioning of an agreed service (step 2), the run-time monitor loads the service SLA from the agreed SLA repository (step 3). Service provisioning is based on the infrastructure resources, which represent the hosts and network resources in a data centre for hosting Cloud services. The resource metrics are measured by monitoring agents, and the measured raw metrics are accessed by the host monitor (step 4). The host monitor extracts metric-value pairs from the raw metrics and transmits them periodically to the run-time monitor (step 5) and to the enactor component (step 6) using our designed communication model.

Upon receiving the measured metrics, the run-time monitor maps the low-level metrics based on predefined mapping rules to form an equivalent of the agreed SLA objectives. The mapping result is stored in the mapped metric repository (step 7), which also contains the predefined mapping rules. The run-time monitor uses the mapped values to monitor the status of the deployed services. In case future SLA violation threats occur, it notifies (step 8) the enactor component for preventive actions. The enactor also receives the predefined threat thresholds (step 8) for possible adjustments due to environmental changes at run-time. This component works out an appropriate preventive action to avert future SLA violation threats based on the resource status (step 6) and defined rules. The enactor's decisions (e.g., *assign more CPU to a virtual host*) are executed on the infrastructure resources (step 9).

3.3.1. Host Monitor

The host monitor processes monitored values delivered by the monitoring agents embedded in the infrastructure resources. The monitoring agents are capable of measuring both hardware and network resources. Figure 3 presents the host monitoring system.

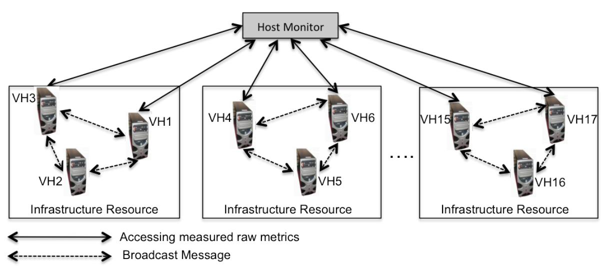

Figure 3. Host Monitoring System

As shown in Figure 3, the monitoring agent embedded in virtual host 1 (*VH1*) measures its resource metrics and broadcasts them to *VH2* and *VH3*. Equally, *VH2* measures and broadcasts its measured metrics to *VH1* and *VH3*. Thus, we achieve a replica management system in the sense that each virtual host has a complete result of the monitored infrastructure. The host monitor can access these results from any virtual host. It can be configured to access different virtual hosts at the same time for monitored values. In case one fails, the result will be accessed from the other. This eradicates the problem of a bottleneck system and offers fault-tolerant capabilities.

3.3.2. Run-Time Monitor

The run-time monitor continuously monitors the customer application status and performance. Its operations are based on two information sources: i) the resource metric-value pairs received from the host monitor and ii) the SLA parameter objective values stored in the agreed SLA repository. The metric-value pairs are low-level entities and the SLA objective values are high-level entities, so for the run-time monitor to work with these two values, they must be mapped into common values.

Mapping of low-level metric to high-level SLAs: As already discussed in Section 3.3, the run-time monitor chooses the mapping rules to apply based on the services being provisioned. These rules are used to compose, aggregate, or convert the low-level metrics to form the high-level SLA parameter. We distinguish between simple and complex mapping rules. A simple mapping rule maps one-to-one from low-level to high-level, as for example mapping low-level metric "*disk space*" to high-level SLA parameter "*storage*". In this case only the units of the quantities are considered in the mapping rule. Complex mapping rules consist of predefined formulae for the calculation of specific SLA parameters using the resource metrics. Table 1 presents some complex mapping rules.

Table 1. Complex Mapping Rules

Resource Metrics	SLA Parameter	Mapping Rule
downtime, uptime	Availability (A)	$A = 1 - \frac{downtime}{uptime}$
inbytes, outbytes, packetsize, avail.bandwidthin, avail.bandwidthout	Response Time (R_{total})	$R_{total} = R_{in} + R_{out} (ms)$

In the mapping rules in Table 1, the downtime variable represents the *mean time to repair (MTTR),* which denotes the time it takes to bring a system back online after a failure situation and the uptime represents the *mean time between failure (MTBF),* which denotes the time the system was operational between the last system failure to the next. R_{in} is the response time for a service request and is calculated as $\frac{Packetsize}{availablebandwidthin - inbytes}$ in milliseconds. R_{out} is the response time for a service response and is calculated as $\frac{Packetsize}{availablebandwidthout - outbytes}$ in milliseconds. The mapped SLAs are stored in the mapped metric repository for usage during the monitoring phase.

Monitoring SLA objectives and notifying the enactor component: In this phase the run-time monitor accesses the mapped metrics repository to get the mapped SLA parameter objectives, which it uses together with the mapped SLAs in the monitoring process to detect future SLA violation threats. This is achieved by comparing the mapped SLA objectives against the threat thresholds. In case of detection it dispatches notification messages to the enactor component to avert the threats. An example of SLA violation threat is something like an indication that the system is running out of storage. In such a case the enactor component acts to increase the system storage.

4. IMPLEMENTATION ISSUES OF THE LOM2HIS FRAMEWORK

The *LoM2HiS* framework implementation targets the fulfilment of some fundamental Cloud requirements such as scalability, efficiency, and reliability. To achieve these aims, the framework is based on well-established and tested open source projects.

4.1. Host Monitor Implementation

The host monitor implementation uses the *GMOND* module from the *GANGLIA* open source project [2] as the monitoring agent. The *GMOND* module is a standalone component of the *GANGLIA* project. We use it to monitor the infrastructure resource metrics. The monitored results are presented in an XML file and written to a predefined network socket. We implemented a Java routine to listen to this network socket where the *GMOND* writes the XML file containing the monitored metrics. Furthermore, we implemented an XML parser using the well-known open source SAX API [5] to parse the XML file in order to extract the metric-value pairs. The measured metric-value pairs are sent to the run-time monitor using our implemented communication model. These processes are repeated periodically.

4.2. Communication Model

The components of our framework exchange large number of messages with each other. So there is a need for a reliable and scalable means of communication. In order to satisfy this need, we designed and implemented a communication model based on the *Java Messaging Service (JMS) API*, which is a Java Message Oriented Middleware (MOM) API for sending messages between two or more clients [6]. In order for us to use *JMS*, we need a *JMS* provider that manages the sessions and queues. We use the well-established open source *Apache ActiveMQ* [7] for this purpose.

Our implemented communication model is based on a queuing mechanism. We use it to realize an inter-process communication for passing messages between two components of the *LoM2HiS* framework, due to the fact that the components could run on different machines at different locations. The queue makes the model highly efficient and scalable.

4.3. Run-time Monitor Implementation

The run-time monitor receives the measured metric-value pairs and passes them into the *ESPER* engine [16] for further processing. *ESPER* is used because the *JMS* system used in our communication model is stateless and as such makes it hard to deal with temporal data and real-time queries. From the *ESPER* engine the metric-value pairs are delivered as events each time their values change between measurements. This strategy drastically reduces the number of events/messages processed in the run-time monitor. We use an XML parser to extract the SLA parameters and their corresponding objective values from the SLA document and to store them in a database. The *LoM2HiS* mappings are realized in Java methods and the returned mapped SLA objectives are stored in the mapped metrics database.

5. FRAMEWORK EVALUATION

We carried out stress tests and performance evaluations as a proof of concept for our framework.

5.1. Evaluation Environment Setup

Figure 4 presents our designed evaluation testbed. The aim of the presented testbed is to test the scalability and performance of the communication model and to produce a proof of concept for the *LoM2HiS* framework. Our evaluation testbed considers one physical host where *GMOND version 3.1.2* is embedded for measuring the resource metric values. From this host we simulate up to 150 virtual hosts. The virtual hosts are simulated with Java threads. Each of the threads becomes a copy of the measured raw metrics from *GMOND*. The host monitor is a Java class running on a different thread. It accesses the measured raw metrics from the virtual host threads, extracts them from their XML files and transmits them as messages (via *Queue In*) into the communication model. The essence of using many virtual hosts is to test the efficiency of the host monitor to process inputs from large number of hosts. This is equivalent to a real environment where the host monitor processes the measured metric-value pairs from different hosts.

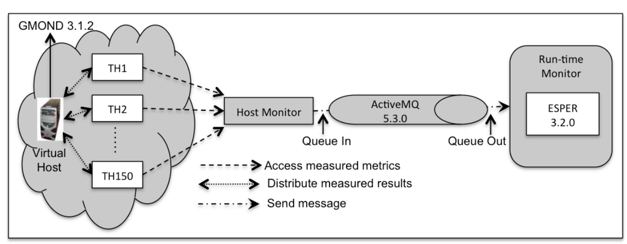

Figure 4. Evaluation Testbed

For the evaluation of the run-time monitor, we defined an SLA agreement for an online web shop as shown in Table 2. The SLA parameter objective values in the table show the quality of service required by the web shop.

Furthermore, in Table 2 we defined the threat threshold values that guide the enforcement of these SLAs.
The used test system consists of an Intel Pentium Core 2 Duo 2.26 GHz, 4GB DDR3 memory, and 3Mb L2 Cache. Mac OS X 10.5 Leopard is the installed operating system and parallel desktop 4.0 is the installed virtualization environment.

5.2. Evaluation Result

This section presents the achieved results of the performance stress test and the performance evaluation using the evaluation setups.

5.2.1. Host Monitor and Communication Model Evaluation Results

Figure 5 presents the evaluation settings and the evaluation results of the host monitor and communication model. For the evaluation settings, four experimental scenarios are defined consisting of number of hosts and number of messages. Each scenario uses one defined queue. As shown in Figure 5, the *y-axis* represents time values and *x-axis* presents the number of hosts used and the number of messages generated and sent through the communication model. The host monitor performance (H_{perf}) is determined considering the three internal functions responsible for: i) measuring the infrastructure resource metrics (*T_measure*), ii) extracting and aggregating the measured metric values (*T_process*), and iii) sending the extracted metric values into the communication model (*T_send*). The overall performance result is then given by the equation: $H_{perf} = T_{measure} + T_{process} + T_{send}$. The communication model performance is equal to the average execution time of the underlying queue (*T_queue*).

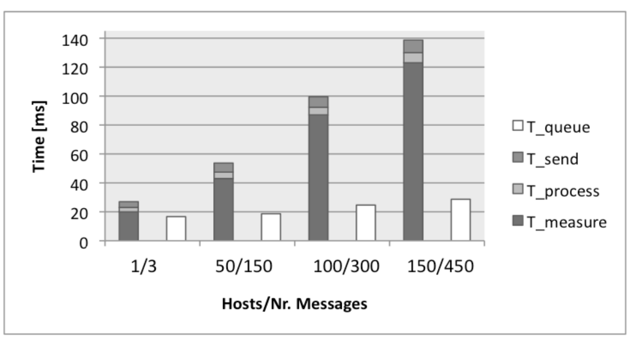

Figure 5. Host Monitor and Communication Model Evaluation Results

From the results presented in Figure 5 it can be noticed in the four scenarios that the host monitor spends most of its time measuring the infrastructure metrics. This shows that this function is critical for the overall performance of the host monitor and will be the point of concentration in our further developments.
The achieved results by the communication model for the different scenarios are relatively stable compared to the number of messages processed.

5.2.2. Run-Time Monitor Evaluation Result

As already discussed in Section 5.1, the evaluation of the run-time monitor is based on the settings presented in Table 2.

Table 2. Run-Time Monitor Evaluation Settings

SLA Parameter	SLA Objective	Threat Threshold
Availability	98 %	98.9 %
Response time	500 ms	498.9 ms
Storage	100 GB	102 GB
Memory	3 GB	3.9 GB
Incoming Bandwidth	50 Mbit/s	52 Mbit/s
Outgoing Bandwidth	100 Mbit/s	102 Mbit/s

The purpose of this evaluation is to test the overall performance of the run-time monitor. Figure 6 depicts the achieved performance result. The *y-axis* represents time values and the *x-axis* the number of hosts. The results are derived from the performances of its core functions responsible for: i) receiving metric-value pairs, passing them into *ESPER engine*, and querying *ESPER* (*T_rec*), ii) extracting the stored SLA from the agreed SLA repository (*T_process*), iii) Applying mappings of low-level metrics to high-level SLA parameters (*T_map*), and iv) monitoring and enforcing agreed SLA objective for services (*T_monitor*). The overall run-time monitor performance (TR_{perf}) is calculated by the equation: $TR_{perf} = T_{rec} + T_{process} + T_{map} + T_{monitor}$.

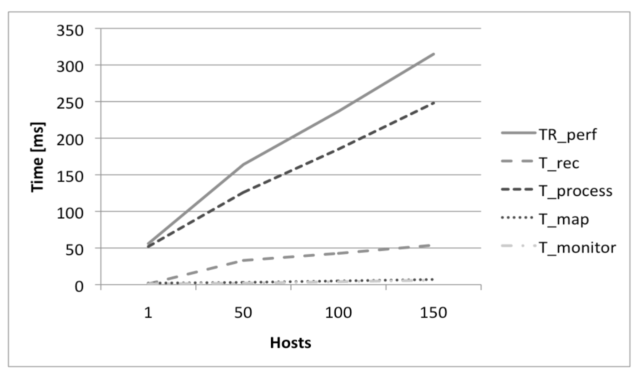

Figure 6. Run-Time Monitor Evaluation Result

According to the results presented in Figure 6 the run-time monitor's overall performance depends highly on the performance of the function to extract the agreed SLA parameters from the SLA repository (*T_process*). We intend to overcome this problem by using decentralized SLA repositories that make them local and fast accessible to each run-time monitor instance monitoring a specific service SLA.

6. CONCLUSION AND FUTURE WORKS

In this paper we presented the *LoM2HiS* framework, which is used for mapping monitored low-level metrics to high-level SLA parameters. The *LoM2HiS* framework is embedded into *FoSII* infrastructure developing infrastructures for autonomic SLA management and enforcement. It is capable of detecting future SLA violation threats based on predefined threat thresholds and can notify the enactor component to avert the threats. We discussed its basic design and implementation. Furthermore, we presented the first experimental results.

Implementing the enactor component is part of our ongoing research work. The design and integration of a graphical interface for the modeling of the metrics using appropriate DSLs is also part of our ongoing research work. Furthermore, we intend to utilize Cloud infrastructure simulation frameworks [19] (e.g., *CloudSim*) to generate manageable, reproducible, and reliable Cloud testing infrastructures.

ACKNOWLEDGEMENTS

The work described in this paper is supported by the Vienna Science and Technology Fund (WWTF) under grant agreement ICT08-018 Foundations of Self-governing ICT Infrastructures (FoSII).

REFERENCES

[1] D. Gunter, B. Tierney, B. Crowley, M. Holding, J. Lee, "Netlogger: a toolkit for distributed system performance analysis", 8th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems, pp. 267-273, 2000.

[2] M.L Massie, B.N Chun, D.E Culler, "Ganglia distributed monitoring system: design, implementation, and experience". *Parallel Computing*, Vol. 30 pp. 817-840, 2004.

[3] R. Buyya, C.S Yeo, S. Venugopal, J.Broberg, I. Brandic. "Cloud computing and emerging IT platforms: Vision Hype, and Reality for delivering computing as the 5th utility". *Future Generation Computer Systems,* Vol. 25(6) pp. 599-616, June 2009.

[4] "Foundation of Self-governing ICT Infrastructures (FoSII)", Available: http://www.infosys.tuwien.ac.at/linksites/FOSII/index.html

[5] "SAX API", Available: http://sax.sourceforge.net/

[6] "Java Messaging Service", Available: http://java.sun.com/products/jms/

[7] "Apache ActiveMQ", Available: http://activemq.apache.org/

[8] B. Koller, L. Schubert, "Towards autonomous SLA management using a proxy-like approach". *Multiagent Grid System*, Vol.3, 2007.

[9] W. Theilman, R. Yahyapour J. Butler, "Multi-level SLA Management for Service-Oriented Infrastructures", 1st European Conference on Towards a Service-Based Internet, 2008.

[10] H.M. Frutos, I. Kotsiopoulos, "BREIN: Business Objective Driven Reliable and Intelligent Grids for Real Business". *International Journal of Interoperability in Business Information Systems,* Issue 3(1) 2009.

[11] "Brein Project" (Business objective driven reliable and intelligent Grids for real business), Available: http://www.eu-brein.com/ 2009

[12] G. Dobson, A. Sanchez-Macian, "Towards Unified QoS/SLA Ontologies". Proceedings of the IEEE Services Computing Workshops (SCW 2006), 2006.

[13] F. Rosenberg, C. Platzer, S. Dustdar, "Bootstrapping performance and dependability attributes of web service", IEEE International Conference on Web Services, pp 205-212, 2006.

[14] A. D'Ambrogio, P. Bocciarelli, "A model-driven approach to describe and predict the performance of composite services", 6th international workshop on Software and performance, pp. 78-89, 2007.

[15] M. Comuzzi, C. Kotsokalis, G. Spanoudkis, R. Yahyapour, "Establishing and Monitoring SLAs in Complex Service Based Systems". IEEE International Conference on Web Services 2009.

[16] "ESPER", Event Stream Processing Engine, Available: http://esper.codehaus.org/

[17] I. Brandic, "Towards Self-manageable Cloud Services", 2nd IEEE International Workshop on Real-Time Service-Oriented Architecture and Applications, Seattle, 2009.

[18] J.O. Kephart, D.M. Chess, "The vision of autonomic computing". *Computer*, Vol. 36:(1) pp. 41-50, Jan 2003.

[19] R. N. Calheiros, R. Buyya, C. A. F. De Rose, "A Heuristic for Mapping Virtual Machines and Links in Emulation Testbeds". 38th International Conference on Parallel Processing, Vienna, pp. 518-525, Sept 2009.

[20] I. Brandic, D. Music, P. Leitner, S. Dustdar. "VieSLAF Framework: Enabling Adaptive and Versatile SLA-Management". 6th International Workshop on Grid Econonics and Business Models 2009.