SLAaaS: an OCCI Compliant Framework for Cloud SLA Provisioning and Violation Detection

Gregory Katsaros, Thijs Metsch and John Kennedy
Intel Corporation, Cloud and Service Lab, Intel Labs Europe, Leixlip, Ireland

Keywords: Service Level Agreements, OCCI, Cloud, Violations, Open Source.

Abstract: SLAs are an integral part of all modern service provisioning operations. They have been a topic of discussion, research and development for many years but still the norm is the use of rigid, complex and not easy to automate Service Level Agreements. In this paper we are presenting a service framework that is leveraging the OCCI specification in order to facilitate standardized SLA provisioning and violation detection. This SLA as a Service (SLAaaS) offering is provided as an open source framework to any Service Provider that wants to efficiently enhance his infrastructure with SLA support.

1 INTRODUCTION

A Service Level Agreement (SLA) is a set of guaranteed terms and conditions that a customer agrees upon related to a service that a Service Provider (SP) is offering. This contractual relationship between the user and the provider of a service is something that both parties appreciate for different reasons. On the one end, the customer demands the existence of a SLA in order to ensure the minimum quality of the service that he receives. On the other side, the provider uses the SLA in order to define and classify the provided offerings, use it as an accounting and billing toolkit or even exploit it for demonstrating the reliability of their provided services.

To this end, cloud SLAs between service consumers and providers about their offerings must be negotiated, stored, modified and monitored in order to ensure the agreed terms and facilitate the service provisioning operations. To this point, a reason that many customers and enterprises are holding off migrations to cloud computing services are the rigid and static SLAs and supported operations that the providers are exposing. These agreements are usually presented as a static text, non-machine readable and non-negotiable (e.g. Amazon EC2's SLA (Amazon, 2015a), S3's SLA (Amazon, 2015b) etc.). Often those contracts are long legal documents or abstract definitions of guarantees that cannot be related with the real needs of the customer.

Such cloud services usually provision a single type of SLAs to all customers, but enterprise users are willing to pay more for individual SLAs with custom terms and more control. On the other hand, some cloud service providers do not have the infrastructure to guarantee their offerings or building such infrastructure is expensive and customized to the technologies they are using. In that context, the OCCI SLA as a Service (SLAaaS) is an open source framework that aims at facilitating the SLA provision process as well as the monitoring and violation detection. It leverages the OCCI (OCCI OGF, 2015) specification and implements an extension which will allow the software defined SLA provisioning.

2 RELATED WORK

There have been in the past several initiatives and effort spend into developing structures and languages that capture the SLA relationship between customers and providers. WSLA (IBM, 2003), WS-Agreement (GRAAP-OGF, 2007) and WS-Policy (W3C, 2007b) are some examples of them that became relatively known in that sector. These languages are focused on the specification of functional and quality properties of web services. WS-Agreement is an OGF recommendation, which is independent from the content of the agreement and does not provide any negotiation features. It is the related OGF specification WS-Negotiation that provides mechanisms for standardized negotiations. These OGF specifications do not cover the establishment of an overarching SLA in multiple service domains though. Other specifications that seek to address explicitly specified semantics include SAWSDL (W3C, 2007a) and OWL-S (DAML-Services, 2008) for service descriptions. Notably an extension for WS-Agreement named Semantic WS-Agreement Partner Selection (SWAPS) has been suggested in (Oldham et al., 2006). On top of all these definitions there have been numerous research projects that built SLA functionalities within service oriented and cloud platforms (e.g. (Georgina Gallizo, 2010) (Mavrogeorgi et al., 2012)), as well as other that tried to contribute towards modelling or machine readable syntaxes (Happe et al., 2011). The report in (Dimosthenis Kyriazis, 2013) captures a big number of European initiatives that dealt with SLAs in manifold ways during the last 10 years.

In the specific case of IaaS, the OGF OCCI working group has suggested a specification on how SLAs can be negotiated and related to IaaS provisioning. This extension to the OCCI has been already proposed and is available for public comment (Gregory Katsaros, 2015) before it becomes a part of the OCCI v1.2 release later this year.

The framework presented in this paper is an implementation of the OCCI SLA specification. It is provided as an open source toolkit through Github[1] under Apache 2.0 license that aims at automating, simplifying and standardizing the SLA provisioning and enforcing process.

3 OCCI SLAaaS FRAMEWORK

SLA as a Service (SLAaaS) is a framework of service components which is offering capabilities of SLA provisioning, SLA template and Service Level Objectives (SLOs) definition, SLA monitoring and agreement violation detection. A Service Provider (SP) can use the framework to setup his own SLA management service or he can collaborate with an independent SLAaaS provider. A service customer can therefore interact with the SLAaaS using proper credentials (related with the SP) using the exposed API in order to discover the available SLA templates, instantiate a SLA and observe whether a term is fulfilled or violated during the service runtime.

The innovations introduced by the SLAaaS are related with the following topics:

- Machine readable and software defined SLAs: through the interfaces exposed by the SLAaaS one could define agreement templates and provision SLAs expressing all the high-level terms as well as the low-level metrics associated with the evaluation of those terms. A structured data format has been introduced to capture those definitions and allow the internal components of the framework to evaluate the terms of each service automatically.
- Standards compliance (OCCI): the interface protocol and data structure of the SLA definitions are compliant with the OCCI and OCCI SLAs extension specification. The standardization of the protocol and interface of the SLAaaS will facilitate the adoption and strengthen the impact of the framework on the cloud community.
- Automated SLA lifecycle process: an important feature of the SLAaaS framework is that it does not only offer provisioning capabilities but a full lifecycle SLA management and violation detection. Upon creation of a SLA resource the monitoring process begins through which the agreement terms are being transformed into policies and actions which are automatically being enforced and triggered.
- Open source and highly configurable software: The overall framework is available as an open source project which with appropriate configuration could serve any kind of service offering and underlying infrastructure.

The SLAaaS framework is a set of components with distinctive responsibilities and operation. The architecture (Figure 1) can be conceptually split into the Provisioning, the SLA Monitoring and Violation Detection and the Data Collection parts. In the following subsections we elaborate on the design and operation of each part of the architecture.

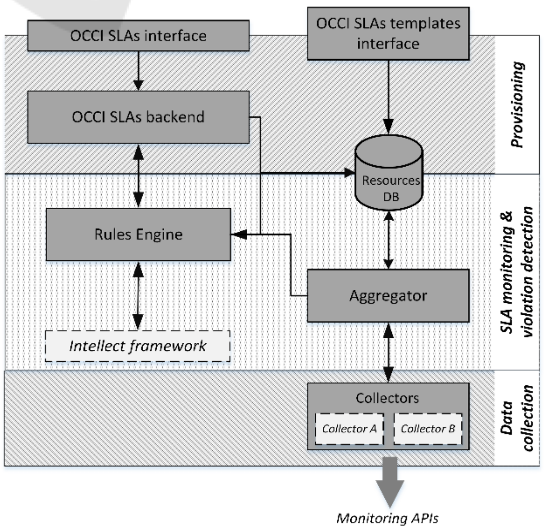


Figure 1: Internal architecture of SLAaaS framework.

[1] https://github.com/IntelLabsEurope/OCCI-SLAs

3.1 OCCI SLA Provisioning

The core capability of the SLAaaS is to provision OCCI compliant service level agreements. To this end, this process follows the rules and principles defined by the OCCI specification (OCCI OGF, 2015). Therefore the framework has an OCCI enabled boundary interface that supports HTTP POST, PUT, GET and DELETE operations for creating, updating, acquiring and removing a SLA resource. The backend structure of the OCCI SLAs has been realized using the Kinds and Mixins mechanism offered by the OCCI specification and has been described in detail in the OCCI SLAs extension specification document. The implementation of the OCCI SLAs extension is based on the open source Python framework Pyssf[2] Our implementation defines all the core entities and mixins of the OCCI SLA mechanism and exposes the necessary RESTful, OCCI enabled interfaces for the management of the resources. In addition, all instances are being stored persistently in a key-value database upon creation.

Table 1: JSON format for definition of a SLA template.

```
{
        "name": "Name of the service provider",
        "scheme": "URI scheme of the service",
        "templates": {
          "name-of-template": {
            "terms": {
              "term-name": {
              "desc": "This is a brief description of the term.",
              "type": "SERVICE-TERM or SLO-TERM",
              "metrics": {
                "metric-1": {
                  "value": X
                }
              }
              "remedy": Y
              }
            }
}
```

An important feature of this SLA framework is the definition of OCCI SLA templates based on the features of a service and the SLOs that a provider wants to introduce. The mechanism that realizes that functionality is based on a custom JSON representation of the SLA template which is being parsed by the SLAaaS and becomes available for the instantiation of OCCI-enabled agreements. In the following table the JSON format for the definition of a template for a service provider is presented (Table 1). The exact structure and words in bold must be used in order for the SLAaaS to effectively parse and introduce the defined SLA template. The rest of the information cover:

- the name of the service provider and URI scheme of the service
- the name of the template
- the terms included in a template the metrics that each term is consisted of
- the remedy represents the cost that the service provider has to pay in case a SLA violation of the specific service and template is being triggered.

Based on that structure, each service can have multiple templates, each template can have multiple terms, and each term can have multiple metrics. As far as the terms definition concerns, the type of one can be either SERVICE-TERM or SLO-TERM. This naming convention follows the suggested structure of the OCCI SLAs specification introduced in the OCCI 1.2. A service term defines the configuration metrics that characterize a service, while a SLO term captures the metrics that define the guaranteed quality of service through a set of metric values. Only the SLO term and its metrics are being monitored during runtime. In order for a SLO term to become violated, all defined metrics of it needs to be violated too. A metric under a SLO term, apart from the value attribute, contains also the attribute limiter type. The limiter type defines whether the declared value is the maximum allowed (limiter type: max), minimum allowed (*limiter type: admin*) or an average value with a certain margin (*limiter type: margin*). In the last case an addition attributed is needed to declare the marginal percentage (*limiter value: X*).

The OCCI SLAs templates interface parses the pre-mentioned structure and creates the appropriate agreement template and agreement terms mixins, which are stored in the persistent database. Those are being later used during the Agreement instantiation process (OCCI SLAs backend).

This OCCI SLA service framework provides a layer of authentication for each request which is achieved through the use of tokens. Initially a user must request a token from the SLA service by providing a username and password which are then validated against a keystone database. If the username and password are valid then a token is returned. This token is then used in subsequent requests to OCCI SLA (Table 2).

Table 2: Example of a request for an authentication token.

```
Curl –i –X GET \
        -H “Authorization:Basic aW9saWU6aW9saWUh” \
        -H “Content-Type:text/occi” \
 ‘https://sla.mobile-cloud-networking.eu/auth/token’
```

The username and password are encoded in base64 and added to the Authorization header. The token typically expires in 50 minutes and a new token will

[2] https://pypi.python.org/pypi/pyssf

need to be generated. In the next sections the authentication token is being passed in a request to create an SLA agreement.

3.2 SLA Monitoring and Violation Detection

The instantiation of a SLA is based on the OCCI protocol and is achieved by an HTTP invocation like the curl command in Table 3.

In the previous example, the SLAaaS is being hosted in the localhost and an agreement of the template medium bronze will be created. The validity period of the agreement is also being specified through the appropriate attributes (effectiveFrom, effectiveUntil). At this point, the SLA instance has been created and there are two additional steps before the monitoring and detection process begins: (a) the agreement instance needs to be associated with the appropriate service resources, and (b) the customer needs to accept the agreement. For the relation of the agreement resource to the service resource an agreement link is being created (Table 4). The contents of the agreement headers are shown in Table 5.

Table 3: OCCI SLA instantiation curl command.

```
curl -i -X POST \
        -H "Category:agreement;
scheme=\"http://schemas.ogf.org/occi/sla#\"" \
        -H "Category: medium_bronze; scheme=
\"http://sla.ran.org/agreements#\"" \
        -H "Content-Type:text/occi" \
        -H "X-OCCI-Attribute: occi.agreement.effectiveFrom =
\"2014-11-02T02:20:26Z\"" \
        -H "X-OCCI-Attribute: occi.agreement.effectiveUntil =
\"2015-11-02T02:20:27Z\"" \
        -H "X-AUTH-TOKEN:
851bff66a097451e84fb39a828ecbccd" \
        -d  'http://localhost:8888/agreement/'
```

Where AGREMENT ID and SERVICE RESOURCE ID are the respective URI identifier of the agreement and service instances. For the acceptance of the SLA and initiation of the monitoring phase the command in Table 6 is being executed.

Table 4: OCCI Agreement Link instantiation curl command.

```
curl -v -X POST localhost.:8888/agreement_link/ -H "$(cat
agreement_headers)"
```

Table 5: Agreement headers for the Agreement Link instantiation.

```
Category: agreement_link;
scheme="http://schemas.ogf.org/occi/sla#"; class="kind"
X-OCCI-ATTRIBUTE: occi.core.source='AGREEMENT_ID'
X-OCCI-ATTRIBUTE:
occi.core.target='SERVICE_RESOURCE_ID'
X-AUTH-TOKEN: 851bff66a097451e84fb39a828ecbccd
Content-Type: text/occi
```

As soon as the SLA instance enters the validity period and is linked (via an OCCI Link) with a resource, the Rules Engine component will identify it, parse all the information related with the defined terms and create a policy file that capture those terms and the appropriate actions to be taken. The time interval which the component searches for valid SLAs is defined within the Rules Engine and represents the time granularity of the SLAaaS operation.

Table 6: Acceptance of OCCI SLA instance.

```
curl -i -X POST \
        -H "Category: accept; scheme =
\"http://schemas.ogf.org/occi/sla#\"" \
        -H "Content-Type:text/occi" \
        -H "X-AUTH-TOKEN:
851bff66a097451e84fb39a828ecbccd" \
        -d
'http://localhost:8888/AGREEMENT_ID?action=accept'
```

The policy mechanism of the SLAaaS system has been developed based on the open source, Python framework Intellect[3]. Using Intellect's syntax, the Rules Engine creates one policy file for each active Agreement instance, which includes one rule for each SLO term in the Agreement template. In order for the Intellect rule to be triggered, at least one of the term's metrics needs to be breached. The Aggregator components takes care of the data acquisition for all the metrics of a SLO term that needs to be assessed.

In case a metric included in a SLO breaches the defined threshold, a violation is triggered by the internal policy mechanism of the SLAaaS Rules Engine. To this end, an OCCI Violation resource is being instantiated to the OCCI SLAs backend and is being linked to the respective Agreement resource. The Rules Engine will continue checking the violated term and as soon as it becomes valid again will restore the state of the term (from violated to fulfilled), remove the Violation instance and continue the normal operation of SLA monitoring and detection. In addition to the creation of the OCCI Violation resource, the Rules Engine registers an event to a message queue in order for any

[3] https://pypi.python.org/pypi/Intellect

actuation component (e.g. infrastructure orchestrator etc.) to be able to detect the violation event. The current implementation supports the RabbitMQ system and the message has the format shown in Table 7.

Table 7: JSON representation of OCCI Violation message.

```
{
     'agreement_id':agreement_id ,
     'timestamp': epoch_timestamp,
     'resource': related_resourece_id,
     'term': term_name,
     'penalty' :remedy_value_from_template
}
```

3.3 Data Collection

SLAaaS is designed to consume different collector for each metric that needs to be monitored. The rationale is that every metrics in an SLO term can be provided by different monitoring systems, thus the API providing the value could vary. In that context, every time that the Aggregator component subscribes a SLO term it invokes the Resources DB in order to acquire the Collector name that each metric is related with. Following that it subscribes each metric against the appropriate collector class. The Collectors components provide the interface definitions for the monitoring collectors. Every metric of a SLO term is being subscribed to the related monitoring API using the limiters threshold defined in the agreement. In case a monitoring system does not provide a subscription/notification mechanism, the collector class will have to raise a thread which will poll the monitoring API for the metrics value and check whether the metrics threshold has been breached. This monitoring interval is defined within the code of each collector and represents the granularity of the violation detection frequency.

4 VALIDATION

4.1 Performance Evaluation

The SLAaaS is a Python-based, WSGI application based on the Pyssf framework for OCCI applications. The internal operation of the mechanism is using threads with a configurable time interval for the detection of the valid SLAs and the monitoring of the terms included in each one of them. In the following charts the performance of the SLAaaS process is being presented in terms of CPU and memory utilization in various frequencies of operation time interval.

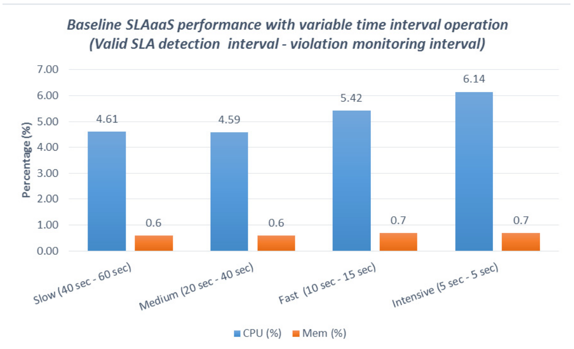


Figure 2: Baseline SLAaaS performance with variable time interval.

Specifically, Figure 2 presents the baseline performance of the SLAaaS when configuring the operational time interval for the valid SLA detection and the violation monitoring to different values. As expected, as the time interval shortens the baseline CPU utilization increases. Overall, the baseline performance, which is defined when hosting less than 10 SLA instances, is relatively stable with a slight peak in the intensive configuration (5 sec - 5 sec).

Being a RESTful, OCCI-based application, the SLAaaS is a highly scalable web application that can provision multiple SLA instances and enable the monitoring capabilities over them. Moreover, the use of python for the implementation results in an efficient execution with small memory and CPU consumption in general. In the following chart (Figure 3) the performance of the framework is being demonstrated while is horizontally scaled by multiple SLA instances.

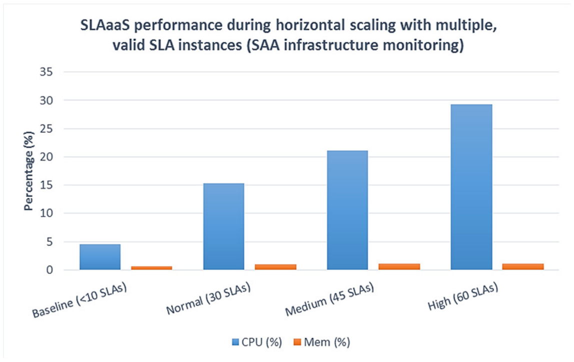


Figure 3: Performance during horizontal scaling using live monitoring data.

[4] At this experiment we used Intels Service Assurance Administrator plugin (http://www.intel.com/content/www/us/en/software/intelservice-assurance-administrator.html).

In that experiment the CPU and memory utilization of the SLAaaS python process is captured while multiple SLA instances are monitored by the framework. It is worth mentioning that the instances are linked with real VM hosted in the Openstack testbed. To this end, the infrastructures monitoring plugin[4] introduces performance constraints that affect the performance of the SLAaaS framework. The variations of response time of the SAA under multiple and simultaneous invocations by the SLAaaS monitoring collector leads to thread execution delay, therefore increased CPU utilization of the python process.

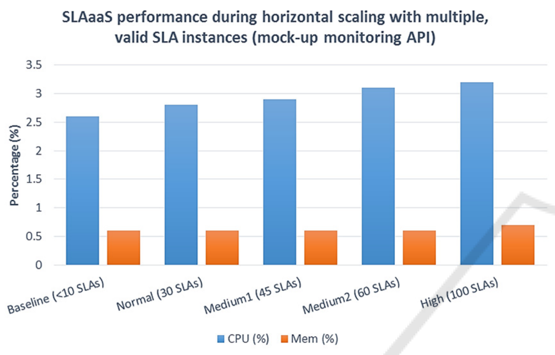


Figure 4: Performance during horizontal scaling using mock-up monitoring API.

In order to evaluate the performance of the SLAaaS without the bottleneck and impact of the monitoring API, we repeated the scalability experiment by replacing the monitoring collector with an efficient mock-up monitoring interface (Figure 4).

The results collected by this experiment demonstrate again the increase of the CPU utilization during the scalability of the framework in a much more gradual rate and with very small captured values. The observed results indicate that the SLAaaS as a standalone application is highly efficient in terms of resources consumed.

This shows normal scalable behaviour which can be effectively managed by allocating more resources (e.g. CPU cores) to the VM instance that hosts the framework. Finally, the SLAaaS python application is affected by the external interfaces that it consumed, specifically the monitoring APIs that it consumes. This is a reasonable finding while the external APIs and interactions cannot be whatsoever controlled or optimized.

The final evaluation experiment of the SLAaaS framework aims at capturing the response time of the RESTfull API and specifically when we request a single resources and when we request all the available resources. This action has been repeated in different SLA workload situations, from a few monitored SLAs (<10) to a high workload of 120 monitored SLAs. In Figure 5 the average response time of 100 GET invocations are being presented, for the single and all of the resources in the running system. A slight increase on the values can be recognised as the number of the monitored load increases. Overall, the values of the response time are very small between one and three milliseconds. The increased response time at the case of the single resources acquisition is because all of the SLA information need to be extracted from the persistent storage, parsed and returned to the client. The GET all invocation returns only the location identifiers of the available resources and not the complete contents of the resources, thus it is being executed quicker.

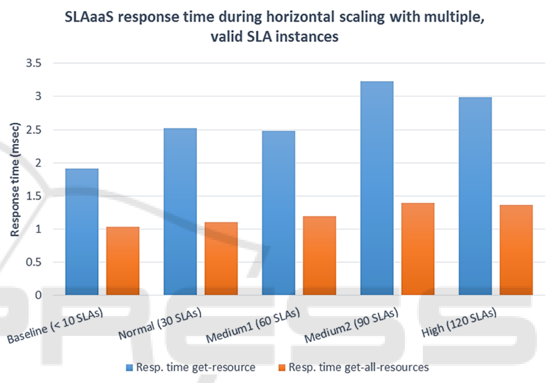
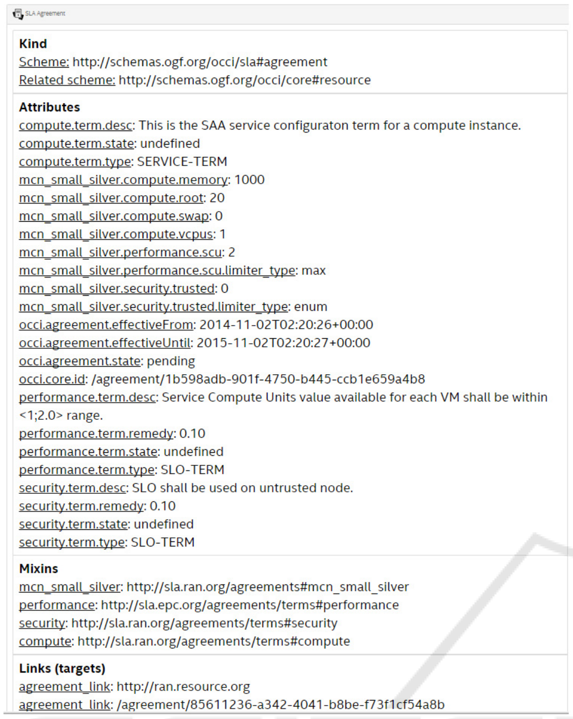


Figure 5: Average response of SLAaaS during horizontal scaling.

4.2 SLA Federation in Multi-Provider Scenario

The ability of effectively capturing the SLA conditions of federated services was a major requirement of the design and implementation of the SLAaaS framework. To this end, the federation of services and consequently of the SLAs that each service provider offers is realized through the Agreement Links feature of the OCCI SLAaaS. Each SLA that is being instantiated has to be linked (with an Agreement Link) with an appropriate resource that represents the offered service. In case of an infrastructure provider such resource could be the compute node identifier or in case of an application/service provider that could be the identifier of a representation of the service instance. In order to realize the federation scenario between two services (e.g. RAN and EPC), an Agreement Link has to be created from the one SLA instance to the other. In Figure 6 the SLA of a RAN service instance is being presented which is linked through an Agreement Link to an EPC SLA instance.



Kind
Scheme: http://schemas.ogf.org/occi/sla#agreement
Related scheme: http://schemas.ogf.org/occi/core#resource

Attributes
compute.term.desc: This is the SAA service configuraton term for a compute instance.
compute.term.state: undefined
compute.term.type: SERVICE-TERM
mcn_small_silver.compute.memory: 1000
mcn_small_silver.compute.root: 20
mcn_small_silver.compute.swap: 0
mcn_small_silver.compute.vcpus: 1
mcn_small_silver.performance.scu: 2
mcn_small_silver.performance.scu.limiter_type: max
mcn_small_silver.security.trusted: 0
mcn_small_silver.security.trusted.limiter_type: enum
occi.agreement.effectiveFrom: 2014-11-02T02:20:26+00:00
occi.agreement.effectiveUntil: 2015-11-02T02:20:27+00:00
occi.agreement.state: pending
occi.core.id: /agreement/1b598adb-901f-4750-b445-ccb1e659a4b8
performance.term.desc: Service Compute Units value available for each VM shall be within <1;2.0> range.
performance.term.remedy: 0.10
performance.term.state: undefined
performance.term.type: SLO-TERM
security.term.desc: SLO shall be used on untrusted node.
security.term.remedy: 0.10
security.term.state: undefined
security.term.type: SLO-TERM

Mixins
mcn_small_silver: http://sla.ran.org/agreements#mcn_small_silver
performance: http://sla.epc.org/agreements/terms#performance
security: http://sla.ran.org/agreements/terms#security
compute: http://sla.ran.org/agreements/terms#compute

Links (targets)
agreement_link: http://ran.resource.org
agreement_link: /agreement/85611236-a342-4041-b8be-f73f1cf54a8b

Figure 6: Attributes list of a federated SLA resource.

The target agreement (in this example the EPC service agreement with ID 85611236-a342-4041-b8be-f73f1cf54a8b) is independent from the source, RAN agreement. It can be instantiated and monitored by the SLAaaS as presented previously. On the other hand, the federated SLA instance (of the RAN service) that incorporates the link to the EPC agreement will begin to be monitored when all the associated SLAs will be accepted and valid (in terms of validity dates). Even if the RAN SLA instance is valid, the RAN-EPC federated service will not start to be monitored unless both of the related SLAs are accepted and valid (shows the log of the SLAaaS when it detects a federation SLA with an invalid related SLA).

As soon as both SLAs are being accepted the SLAaaS will subscribe the included terms to the monitoring system and begin the detection and evaluation process. During the latter, when a violation is detected in a term of one of the two related SLAs then the complete federation SLA is considered to be violated.

In Figure 7 and Figure 8 the OCCI Viz[5] application

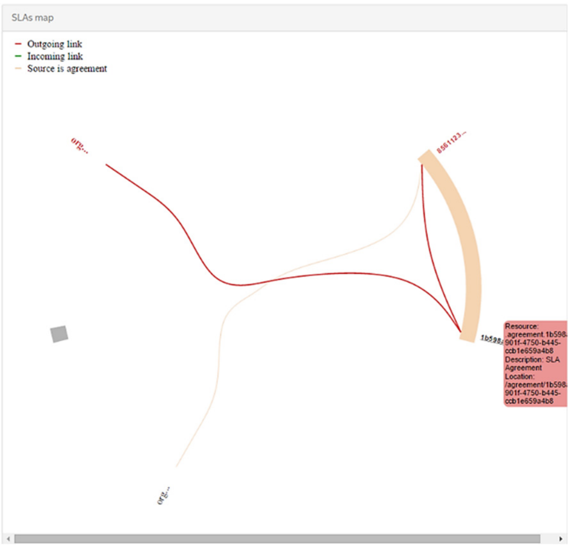

SLAs map
Outgoing link
Incoming link
Source is agreement
org...
8561123...
1b598...
Resource: .agreement.1b598... 901f-4750-b445-ccb1e659a4b8
Description: SLA Agreement
Location: /agreement/1b598... 901f-4750-b445-ccb1e659a4b8
org...

Figure 7: Representation of the federated SLA with two outgoing links towards another SLA and a resource (e.g. compute).

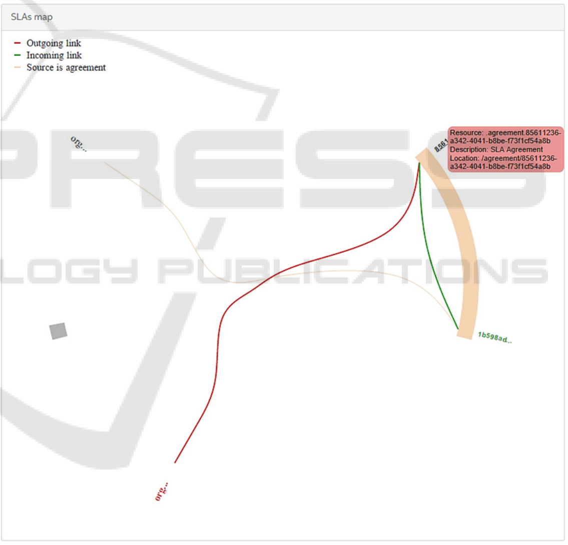

SLAs map
Outgoing link
Incoming link
Source is agreement
org...
8561
Resource: .agreement.85611236-a342-4041-b8be-f73f1cf54a8b
Description: SLA Agreement
Location: /agreement/85611236-a342-4041-b8be-f73f1cf54a8b
1b598ad...
org...

Figure 8: Representation of an SLA with an outgoing link towards a resource (e.g. compute) and an incoming link from an SLA resource.

is used to demonstrate the relation of the federation SLA. In circular graph the SLA entities are represented as points in the coloured arc. Those points are linked through a coloured line to other resources such as infrastructure nodes, services or other SLA instances. The direction of the links and therefore the service federation is demonstrated through the different colours of the depicted links. With red colour an outgoing link is presented while with green colour an incoming link.

[5] https://github.com/IntelLabsEurope/OCCIViz

5 CONCLUSIONS

In any kind of service provisioning operation Service Level Agreements are and will the means for guaranteeing the quality of the offered service. The Software as a Service (SaaS) revolution will only highlight the immediate need for effective and easy to use SLA assurance systems. In addition, the developments in Software Defined Infrastructures (SDI) demand the automation of all features within the service provisioning and deployment.

In that context, the SLAaaS framework presented in this paper is an effective, open source solution that offer SLA and SLA templates provisioning as well as violation detection features. It is based on the OCCI specification, thus, exposes a simple but standardized boundary interface over common HTTP RESTful operations. It is configurable in order to be plugged into any monitoring API by adding custom collectors' implementations. The successful instantiation of an Agreement resource leads to the enabling of the SLA monitoring process which automatically detects any violation, instantiates a respective OCCI resource and notifies a message queue for that specific event.

Overall, the SLAaaS is a powerful toolkit that can be easily integrated with any type of infrastructure and enhance the operations and offerings of a Service Provider.

ACKNOWLEDGEMENTS

This work has been supported by the Mobile Cloud Networking project (MCN, 2015) has been partly funded by the European Commission's ICT activity of the 7th Framework Programme (FP7/2007- 2013) under contract number 318109 as well as by the CloudWave FP7 project (contract number 610802) (CloudWave, 2015).

REFERENCES

Amazon (2015a). Amazon EC2 Service Level Agreement. http://aws.amazon.com/ec2-sla/.

Amazon (2015b). Amazon S3 Service Level Agreement. http://aws.amazon.com/s3-sla/.

CloudWave (2015). CloudWave FP7. http://cloudwavefp7.eu/.

DAML-Services (2008). DAML Services. http://www.daml.org/services/owl-s/.

Dimosthenis Kyriazis (2013). Cloud computing. SLAs Expoitation of research results. http://ec.europa.eu/newsroom/dae/document.cfm?doc id=2496.

Georgina Gallizo, Roland Kbert, G. K. K. O. K. S. S. G.

E. O. (2010). A service level agreement management framework for real-time applications in cloud computing environments. *In Proceedings of the 2nd International ICST Conference on Cloud Computing*, Cloud Com 2nd, pages 26–28, Barcelona, Spain. ACM.

GRAAP-OGF (2007). Web Services Agreement Specification. http://www.ogf.org/documents/GFD.107.pdf.

Gregory Katsaros (2015). OCCI SLAs extension specification. https://redmine.ogf.org/issues/276.

Happe, J., Theilmann,W., Edmonds, A., and Kearney, K. T. (2011). *Service Level Agreements for Cloud Computing*, chapter A Reference Architecture for Multi-Level SLA Management, pages 13–26. Springer New York, New York, NY.

IBM (2003). IBM Web Service Level Agreements. http://www.research.ibm.com/people/a/akeller/Data/WSLASpecV1-20030128.pdf.

Mavrogeorgi, N., Gogouvitis, S., Voulodimos, A., Katsaros, G., Koutsoutos, S., Kiriazis, D., Varvarigou, T., and Kolodner, E. K. (2012). Content based slas in cloud computing environments. *In Proceedings of the 2012 IEEE Fifth International Conference on Cloud Computing, CLOUD '12*, pages 977–978, Washington, DC, USA. IEEE Computer Society.

MCN (2015). Mobile Cloud Networking FP7. https://www.mobile-cloud-networking.eu/.

OCCI OGF (2015). Open Cloud Computing Interface Core. https://www.ogf.org/documents/GFD.183.pdf.

Oldham, N., Verma, K., Sheth, A., and Hakimpour, F. (2006). Semantic ws-agreement partner selection. *In Proceedings of the 15th International Conference on World Wide Web, WWW '06*, pages 697–706, New York, NY, USA. ACM.

W3C (2007a). Semantic Annotations for WSDL and XML Schema. http://www.w3.org/TR/sawsdl/.

W3C (2007b). Web Services Policy 1.5 Framework. http://www.w3.org/TR/ws-policy/.