rSLA: Monitoring SLAs in Dynamic Service Environments

Heiko Ludwig[1(✉)], Katerina Stamou[1], Mohamed Mohamed[1], Nagapramod Mandagere[1], Bryan Langston[1], Gabriel Alatorre[1], Hiroaki Nakamura[1], Obinna Anya[1], and Alexander Keller[2]

[1] IBM Research - Almaden, San Jose, CA, USA
{hludwig,stamou,mmohamed,pramod,bryanlan,galatorr, obanya}@us.ibm.com, hnakamur@jp.ibm.com
[2] IBM Global Technology Services, Chicago, IL, USA
alexk@us.ibm.com

Abstract. Today's application environments combine Cloud and on-premise infrastructure, as well as platforms and services from different providers to enable quick development and delivery of solutions to their intended users. The ability to use Cloud platforms to stand up applications in a short time frame, the wide availability of Web services, and the application of a continuous deployment model has led to very dynamic application environments. In those application environments, managing quality of service has become more important. The more external service vendors are involved the less control an application owner has and must rely on Service Level Agreements (SLAs). However, SLA management is becoming more difficult. Services from different vendors expose different instrumentation. In addition, the increasing dynamism of application environments entails that the speed of SLA monitoring set up must match the speed of changes to the application environment.

This paper proposes the rSLA service and language that is both flexible enough to instrument virtually any environment and agile enough to scale and update SLA management as needed. Using rSLA the time of setting up SLA compliance monitoring of application environments involving infrastructure, platform, and application services can be significantly reduced.

Keywords: Service Level Agreement · Cloud Computing · PaaS · Monitoring · Reporting

1 Introduction

Cost, availability and on-demand scaling have accelerated the adoption of different application deployment models. Today's application environments combine Cloud and on-premise infrastructure as well as platforms and services from different providers to enable the quick development and delivery of solutions to their intended users. For example, a Web or mobile application of an online

A. Barros et al. (Eds.): ICSOC 2015, LNCS 9435, pp. 139–153, 2015.
DOI: 10.1007/978-3-662-48616-0_9

merchant might be deployed on a cloud-based Platform-as-a-Service (PaaS), use persistence services within the platform, run ID and login management through a third-party Web service, and be monitored by various in-house logistics systems.

The ability to use Cloud platforms to stand up applications in a short time frame, the - now - wide availability of services, and the application of a continuous deployment model has led to a much more dynamic application environment, changing at high velocity.

Managing quality of service has become more important in the context of the Cloud and the prolific use of micro-services. The more external service vendors are involved the less control an application owner has over the quality of the delivery of this service and must rely on the quality commitments of his or her vendors in the form of Service Level Agreements (SLAs).

SLAs are widely used in the context of corporate Information Technology (IT) outsourcing. Practically every outsourcing customer requires his or her vendors to commit to SLAs and holds vendors accountable with financial penalties or rewards for achieving objectives. Objectives typically relate to the performance of systems and services, their availability, and the performance of service processes such as provisioning servers and responding to help desk tickets. Many vendors sell SLA management systems, often as part of service management suites (e.g., IBM Integrated Service Management/ISM) or as Software-as-a-Service (e.g., ServiceNow). In enterprise practice setting up SLA management for a particular domain and monitoring compliance on an ongoing basis requires substantial integration projects to collect metrics in a data warehouse. Aggregating and adjudicating SLA compliance likewise often requires substantial manual work or ad-hoc scripting.

While this is viable for large outsourcing contracts and all their intricate details this is not a good match for today's dynamic application environments using Clouds and services from multiple vendors. One key problem is the heterogeneity of interfaces. Services from different vendors have different instrumentation and use different service management systems making it difficult to collect and aggregate performance data for service level objective evaluation. While system vendor heterogeneity is certainly an issue within one enterprise it is easier to avoid and system level standards such as the Common Information Model (CIM) of the Distributed Management Task Force (DMTF) help management systems to interact with servers, network components and the like. Management APIs on a Cloud and service level have not yet undergone such widely accepted standardization although some schemes have been proposed [6].

In addition to heterogeneity, the increasing dynamism of application environments entails that SLA monitoring must be set up at the speed of change of the application environment. Continuous deployment, or DevOps, enables constant changes to applications and the binding to new services. Cloud infrastructure and - in particular - platform services enable the deployment of new applications on very short notice. Organizations can respond rapidly to novel needs and change their application environment at a fast pace.

To provide effective SLA management in a Cloud environment, the SLA management system must be able to set up the monitoring of customer-specific SLA terms against a heterogeneous set of service instrumentation in a short amount of time.

This paper proposes the rSLA service, which is both flexible enough to instrument virtually any environment and agile enough to scale and update SLA management as needed. Given an SLA expressed using a formal SLA specification, rSLA sets up the monitoring infrastructure and starts monitoring compliance. Using rSLA the time of setting up SLA compliance monitoring of application environments involving infrastructure, platform, and application services can be significantly reduced and aligned with a typical DevOps life-cycle.

The remainder of this paper is organized as follows: The next section discusses related work. Subsequently, we give an overview of the approach, followed by a discussion of the rSLA language. After that we outline architecture and execution model, including the implementation. In this section we also show in a case study how we applied our approach to a real customer environment. Finally, we summarize and conclude.

2 Related Work

QoS has been an important part of IT Service management for a long time. As a result, a significant body of work has focused on different aspects of QoS - from measurement to enforcement.

In addition to typical IT service SLA management products mentioned in the introduction, many providers of Cloud and networking services do provide specific service performance/availability information on their Web sites or through APIs. This is often the basis of tiered service offerings, with better quality demanding a higher price. For instance, Amazon provides a generic monthly uptime SLO of 99.95 % for their compute resources and performance SLO of 2 k IOPS for EBS SSD volumes at the point of writing of this paper. [2] provides a survey on current service level provisioning practices and compares the SLAs of five public cloud service providers. While this typically entails some kind of availability/performance guarantees it is not a custom SLA, tailored to a client's business needs. This one size fits all policy is often too restrictive.

An important part of a solution to automate SLA management is a formal, machine-interpretable representation of the SLA. Several specifications have been proposed in the Web service and Grid context such as the Web Service Level Agreement language (WSLA) [10], the Web Services Offer Language (WSOL) [17] and Web Service Agreement (WS-Agreement) standard of the Open Grid Forum [1]. WSLA is designed to capture SLAs for web services and is used, among other things, to facilitate automatic service configuration, e.g., to configure load balancers of clustered Web servers. WS-Agreement inherits language characteristics of the WSLA specification and has been used in numerous initiatives [3,16]. SLA representations were also used for on-demand service provisioning [5,9] and automated service composition [18].

These uses require the automatic set up of SLA compliance monitoring for the systems they are applied to, e.g., Web services. While reading metrics at specific points of a distributed cluster could be described in some representations such as WSLA there was no prescribed way to address the heterogeneous instrumentation found in today's Cloud scenarios. It was not really necessary because the systems instrumented were assumed to be Web servers, Grid schedulers, etc. A number of approaches that use SLAs as a key element of compliance evaluation, such as [4,13,14], have been proposed for application monitoring in dynamic systems. Like these authors, we focus on comprehensive QoS monitoring of SLAs. However, our approach includes a flexible and scalable monitoring mechanism and action framework required for quick setup of SLA monitoring in today's heterogeneous and dynamic cloud environment.

Recent work has investigated SLAs for Cloud deployments. In [7] Kouki et al. propose the Cloud Service Level Agreement language (CSLA) that enables the definition of SLAs for any type of cloud service. CSLA is intended for dynamic service provisioning environments and is based on the Open Cloud Computing Interface (OCCI) [15] and the Cloud Computing Reference Architecture of the National Institute of Standards and Technology (NIST) [8]. While these proposals define some common metrics and SLA parameters to set individually they reflect more an advertised QoS than a commitment to an individual customer.

In contrast, our work focuses on building a light weight, extensible system for SLA management with the goal of rapid deployment and adaptability to ever changing cloud landscape and customizability to different client business requirements.

3 Overview of the Approach

The key objective of our SLA compliance monitoring system presented in this paper is the fast setup of monitoring an SLA in an environment in which the subject of monitoring is evolving quickly and services expose heterogeneous measurement interfaces. This section provides an overview of our approach how to address all of those issues at the same time. The first subsection discusses the system model outlining how the rSLA service interacts with interfaces of a heterogeneous environment. The second part describes the major elements of an SLA document's content.

3.1 System Model

Cost, availability and on-demand scaling have accelerated customer adoption of different application deployment models - on-premise Cloud, private Cloud, public Cloud or hybrid Clouds, in addition to their traditional dedicated environments. Depending on the services an application depends on, the choice of platform varies. A single application could potentially be reliant on multiple Clouds for the set of services it consumes. From a customer or tenant perspective, monitoring service levels that their workloads are receiving often becomes a complex task of aggregating information across multiple Cloud providers, each

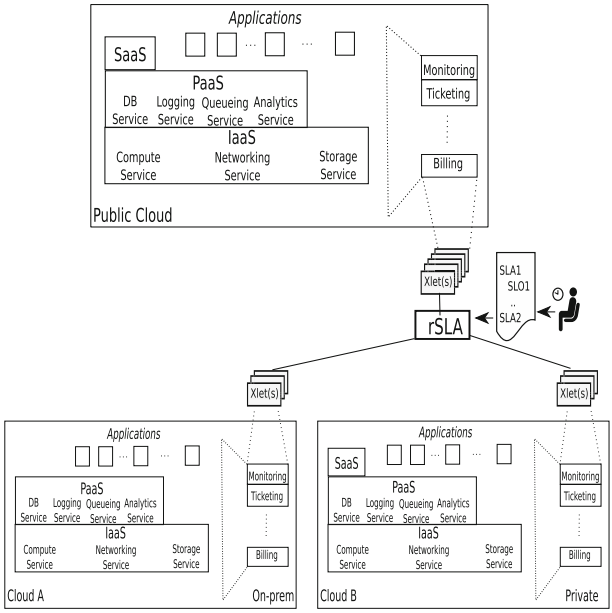

Fig. 1. rSLA system overview

using different proprietary monitoring and management stacks. Furthermore, the DevOps transition increases the rate of change of applications and consequently their bindings to services.

Figure 1 shows an overview of the proposed rSLA system model. The rSLA framework is made up of three main components:

1. a language for administrators or service providers to express service level agreements;
2. a set of Xlets - lightweight, dynamically bound adapters; Xlets abstract the heterogeneity of service management interfaces to rSLA, both for the reading of metrics as well as the acting on the result of SLA evaluation; and
3. a modular SLA evaluation service; this service interprets the rSLA document. It performs the reading of measurements through monitoring Xlets as defined in the document, aggregates metrics, evaluates compliance and then notifies stakeholders.

A formal language for SLAs is the key to both custom SLAs on a case-by-case basis and fast deployment of the SLA. The Xlet approach provides the level of abstraction and indirection to intermediate between service interface heterogeneity and the need of the SLA evaluation service to read metrics in a homogeneous way.

The rSLA evaluation itself can be deployed in any of the Cloud deployment models - on-premise Cloud, private Cloud, public Cloud or hybrid Clouds. Further sections describe the language concepts, the execution model, and implementation. The case study later in the paper discusses some Xlets we wrote for a particular Cloud service.

3.2 SLA Model

An rSLA document describes how metrics are to be obtained from instrumentation and how they are to be aggregated. Based on those metrics Service Level Objectives (SLOs) can be defined and it can be specified how to proceed if those SLOs are met or violated. Figure 2 illustrates the main concepts of the proposed approach using a simple example.

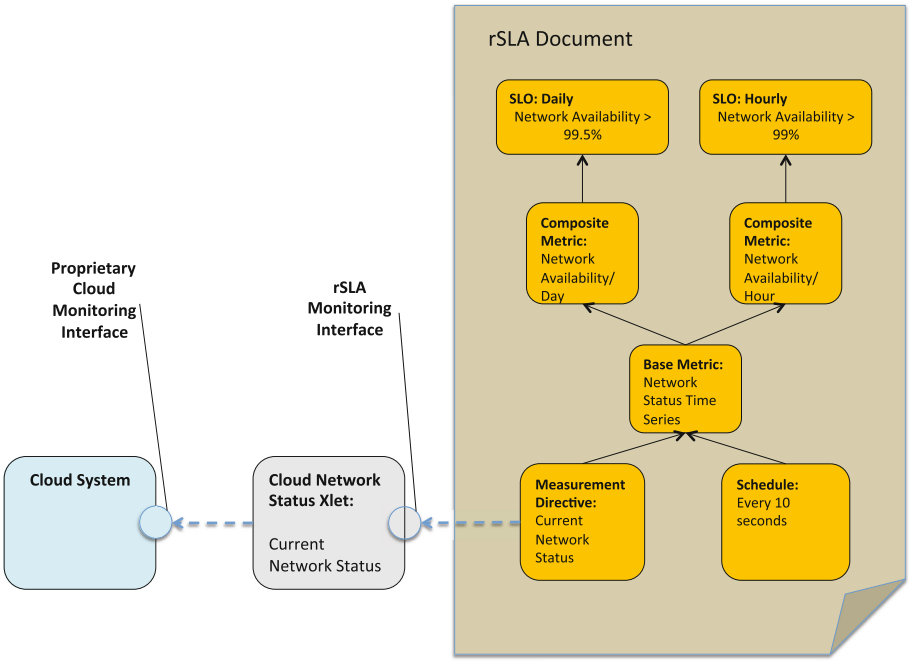

Fig. 2. rSLA concepts overview

This example describes an SLA defining networking guarantees. The customer is interested in a daily guarantee, which needs to exceed 99.5 % availability, and an hourly one, which needs to be only 99 %. The current status of the network at any given point in time can be obtained at the proprietary interface of the Cloud provider with whom the customer enters the SLA. There is an Xlet exposing an rSLA monitoring interface that can be requested to provide network status in an rSLA standard way and which in turns requests this status from the proprietary Cloud management interface.

When expressing the rSLA document the author of this document has to describe how to translate what can be read by instrumentation into a high-level metric on which the customer wants to have a commitment. In addition, the actual SLO must be articulated. This gives us our main set of concepts for rSLA:

The foundation of measurement are *base metrics*. Base metrics describe how metrics are to be obtained from instrumentation and how frequently. For this purpose a base metric has a measurement directive and a schedule.

The *measurement directive* describes how a specific metric can be obtained through an Xlet. It points to an Xlet, the specific service entity, and the metric to be obtained. For example, our networking Xlet for this Cloud provider may provide status for an external access network as well as the internal network, connecting the different virtual machines of this customer (service entities). Multiple metrics may be available for those entities such as availability and latency. The measurement directive will point to the specific metric of a specific entity.

The *schedule* describes the frequency at which measurements are taken. The periodicity can reflect the customer's needs. For our networking example this time frame will typically be a sampling interval in a range of seconds to minutes.

Composite metrics aggregate values of base metrics and other composite metrics. The aggregations are described using expressions over values from the metrics it depends on. The phrasing of the expression must be type-compatible with the types of the input metrics. Oftentimes, expressions involve aggregations of time series, e.g., to averages over a time period, or aggregate different metrics such as the network availability of multiple network segments. In our example, two composite metrics depend on the same base metric, one aggregating network state in the window of the last hour, one in the window of a day. While the basic approach to metric aggregation is simple, we often find quite complex functions excluding some values and performing complex stochastic operations. This puts significant requirements on the richness of expressions the language provides.

Service Level Objectives define the commitment of a service provider to its customer. This is defined in a Boolean expression over metric values. Oftentimes, commitments are bounded by a precondition. For example, a response time guarantee for a REST call is only given if the request rate to the service is less than a certain number of calls per minute. This is commonly used to scope the applicability of the SLO. SLOs can also be associated with a schedule that defines when to evaluate the precondition and objective expressions. Alternatively, it can be evaluated each time input metrics have new values. This might lead to a significant computation effort when large numbers of metrics are sampled at high frequency.

Finally, though not shown in the figure, *notifications* define how external services are to be informed about the state of SLOs and metrics. The rSLA language provides notifications as event-condition-action rules that describe when notifications should take place, according to a schedule or when a new evaluation is available; the specific condition, e.g., upon violation; and how to notify. For this "how" part, the action, we fall back to our Xlet concept. Notification Xlets provide a homogeneous interface abstracting from various possible recipient interfaces. Notification directives describe how to connect to those Xlets. Notification statements also include a description which information is to be passed on, the marshalling of the action.

Based on this small set of concepts SLA authors can express a large variety of SLAs. Measurement directives connect the rSLA language to the systems model; composite metrics can express custom metrics aggregation to the level needed by the client; SLOs express the commitment; notifications define how to actively communicate the SLO status and other information to recipient services.

4 rSLA Language

The rSLA language allows an SLA author to express the concepts outlined in the previous section. The specific language design is key to the consumability of the language and the usability of the approach as a whole. Prior to diving into the specifics of our language elements we want to briefly discuss some issues and guiding principles.

4.1 Design Considerations

Many of past and current approaches to representing SLAs such as WSLA, WS-Agreement, WSOL and CSLA choose XML as a substrate. While some of those languages have enjoyed success in systems deployment or as basis for further research work none has seen wide-spread adoption in industry at this point. While not having systematically studied adoption, the authors of this paper have been involved in some of those efforts and received feedback as to why or why not to use one language or another. A key item of feedback from practitioners was that XML is hard to read. While there is always the opportunity to provide advanced editors to not expose authors to the actual representation many professionals actually prefer writing system-level scripts.

When considering who actually writes rSLA documents it turns out that this would often be administrators with significant experience in scripting. The rSLA syntax is designed to resemble other languages that administrators use on a daily basis. Many domain specific languages have seen acceptance lately, e.g. Chef (www.chef.io). rSLA aims at providing a similar experience to its authors.

Another issue with early efforts such as WSLA is the limitation of its expression language, or its complete absence in the normative specification, in case of WS-Agreement. While WSLA has an extensible set of functions, a practitioner would actually have to change its evaluation engine to extend it. This is not practical. WS-Agreement suggests to use the expression language of your choice, again requiring an addition to a runtime system. From a practitioner's perspective they are incomplete. The rSLA language is meant to encompass a wide set of functions that can be used in expressions. In addition, the language design enables to define extension to the standard language expression scope as part of the SLA document itself.

To achieve those objectives the rSLA language is designed as a Domain Specific Language (DSL) on the basis of Ruby - hence the "r" in rSLA. Ruby, as a substrate language, has some characteristics that makes it suitable as a

basis of a DSL, in particular its scripting approach and its easy access to meta-interpretation. Many DSLs such as Chef are also based on Ruby and administrators often have encountered those before. Other languages and their interpreters might be equally suitable from a technical requirements perspective.

4.2 rSLA Language Elements

This section discusses the rSLA language elements in an overiew and example manner. We provide a small rSLA document that specifies three rSLA objects: an SLA, a base metric and a service level objective. The SLA can be read by an rSLA engine in a cloud runtime environment. A full specification of the language is beyond the scope of an individual paper of this format. The rSLA language documentation describes in detail the language structure, the use of rSLA statements, and its production rules [11].

Listings 1.1 and 1.2 illustrate example statements for the creation of an SLA and a base metric and, respectively, of an SLO object. A deployed rSLA service can read and process these statements in a runtime environment. An rSLA document must have exactly one *sla* statement. The *sla* statement defines the parties to the SLA.

```
1  sla do
2    tenant "ExampleClient"
3    provider "ExampleProvider"
4  end
5
6  basemetric do
7      name "bareMetalProvisioning"
8      unit "seconds"
9      type "event_set integer"
10
11     measurementdirective do
12         entity "/process/baremetal_provisioning"
13         type "event_set integer"
14         source "http://provisioningxlet.stage1.mybluemix.net/
               process/baremetal_provisioning/time"
15     end
16
17     schedule do
18         frequency "1"
19         unit "m"
20         method "every"
21     end
22 end
```

Listing 1.1. rSLA SLA (lines 1–4) and basemetric (lines 6–19) statements

The base metric definition includes its *measurementdirective* and may include a *schedule* if values are received according to a schedule. Its attributes are *name*, *unit* and *type*.

The type of this particular example is an *event_set* of *integers*. Types of metrics can be either simple types or one of two complex types: *time_series* and

event_set. An event set is a set of values collected over time. Typical examples include provisioning events. In a given time interval any number of provisioning processes may have occurred. If a provider wants to give a guarantee over the time it takes to provision, say for 90 % of the processes, we need a complex type able to accommodate this set of values representing the time it took. Time series, on the other hand, are equidistant readings of values.

The measurement directive describes where to read the base metric values. It describes the entity in question, its type, and the source. In this specific case the source is a fully articulated URL. Based on the detailed metric name and the entity specification a reference to an Xlet name could have sufficed as well.

The schedule follows the syntax of the Rufus scheduler [12], which is sufficient for our purposes.

Another required element of an SLA is the SLO. An example of such an SLO statement is illustrated by Listing 1.2.

```
slo do
    name "bareMetalProvisioning"
    precondition "bmtotalProvisionsNumber.value≤100"
    objective "bmGoodProvisionsNumber.value\
        bmtotalProvisionsNumber.value≥0.9"

    schedule do
       frequency "60"
       unit "m"
       method "every"
    end
end
```

Listing 1.2. rSLA SLO creation script

The SLO definition in Listing 1.2 provides an example of an SLO statement in the rSLA language. The *slo* has a name to refer to, a schedule according to which it is evaluated and two expressions: the *precondition* defines the bounding condition of the SLO while the *objective* defines what must be achieved.

Upon SLO evaluation, the rSLA engine will first evaluate the precondition block. If the logical outcome from the execution of the precondition block is false, the SLO is met. In our case, if the total number of bare metal servers provisioned in the past hour is less or equal than hundred, the SLO applies. If it is larger, it doesn't - and no violation occurs. In case the precondition is true or if there is no precondition block, the rSLA runtime evaluates the objective block. If the logical outcome from processing the statements in the objective block is true the SLO is healthy. Otherwise the SLO evaluation indicates a violation.

Both base metric and SLOs enable a user to define schedules for the measurement and respectively the evaluation of such rSLA instances. Schedule details are passed to a scheduler service that instantiates schedule objects and coordinates their processing.

Both precondition and objective are expressions following Ruby syntax. Values of metrics, both base metrics and composite metrics, are referred to by

their name "." value. This can refer to a complex type or a simple type. Ruby expressions resolving to a Boolean type are valid. This enables authors to use any available Ruby function, providing the full expressiveness of the substrate scripting language.

Not illustrated for space reasons is the *composite_metric* statement. It has name, unit and type like the *base_metric* but rather than a measurement directive it has an expression as well, which has to yield a result according to its type. Equally, all type-compatible Ruby expressions are valid.

This syntax allows administrators and others of similar skill to define SLAs in a scripting-like DSL. Using an existing scripting language as substrate, in our case Ruby, enables us to use a full scope of expressions.

5 rSLA Runtime Architecture and Implementation

The rSLA Service is designed to be run either by a service provider, a customer, or a third party such as a service integrator. The following section explains the components constituting the rSLA service and its interaction with xlets.

5.1 rSLA Service

The rSLA service is a Ruby application, a given because it is a Ruby DSL. rSLA offers different REST interfaces that allow the management of the life cycle of an SLA described using our DSL.

The rSLA service exposes a set of life cycle management functions to its user, in particular to create, activate, deactivate and remove SLAs. At the reception of a new SLA, the rSLA Service interprets the file and creates a set of objects. The new objects are persisted in a Cloudant no-SQL database service using a CouchRest data model.

On the *activation* of an SLA, the rSLA Service orchestrates all the needed operations to activate and manage the SLA life cycle. It starts by scheduling data collection for base metrics with the rSLA scheduler service, which is a companion service to the rSLA application itself. Based on the defined schedules, the scheduler triggers rSLA Service interfaces exposed to the scheduler associated with metrics. The rSLA service then invokes the monitoring Xlets to collect new observations for the related base metrics. The observations received from Xlets are persisted in Cloudant. Observations are JSON structures returned by monitoring Xlets that contain the metric value requested, a time stamp and an *extras* object that contains more information useful for further diagnostics. For example, in case of our provisioning processes, in addition to the provisioning time as the value of the metric it may be interesting to know which particular server has been provisioned if a provisioning process takes longer than expected.

Similarly, according to the schedule of an SLO, the Scheduler triggers an rSLA Service interface for SLO evaluation. The rSLA service then retrieves the data related to the specific SLO, reifies the corresponding metrics as Ruby

objects and then evaluates the expressions of the SLO, precondition and objective. As an optimization step, this evaluation may use the map-reduce functions offered by Cloudant to delegate possible parallel processings to Cloudant and benefit from its efficiency.

Afterwards, the rSLA Service generates JSON notifications representing the results of the evaluation. These notifications are sent to the Notification Xlet for formatting and reporting to the client.

5.2 rSLA Xlets

As described previously, Xlets offer a standard interface for monitoring and notification as a generic REST API, abstracting from the heterogeneity of different service interfaces. In our architecture, Xlets are services on Bluemix. An Xlet is customized according to its role in the overall system. As shown in Fig. 3, each Xlet provides three interfaces:

- *CFBrokerInterface*: Since the Xlets are provided as services on Bluemix, they need to offer this generic interface that describes exactly how to provision the service, how to unprovision it, how to bind the service to a given application and how to unbind it.
- *ConfigurationInterface*: In order to ensure multi-tenancy and customization of an Xlet, it must offer an interface to configure its tenancy. This interface could offer other functionalities of customization, e.g., to configure the access credentials for Cloud resources.
- *RuntimeInterface*: This interface provides the main business of the xlet. It describes the specific functionalities to be offered by the application instance (e.g., monitoring services, reporting services).

All Xlets observe the same architecture but differ in their implementations from one use case to another. The runtime interface of the monitoring Xlets are in line with the DMTF Cloud Infrastructure Management Model [6]. They allow collecting monitoring data for a specific type of resources with different granularities. For example, Fig. 3 shows a SoftLayer specific Xlet. The SLXlet allows to get monitoring data of SoftLayer-provisioned servers for a given account. The account credentials are passed to the Xlet in the configuration phase through the Configuration interface. Afterwards, the runtime interface of the Xlet can be used to get the list of servers, the list of metrics for a given server, or the value for a given metric for a specific server.

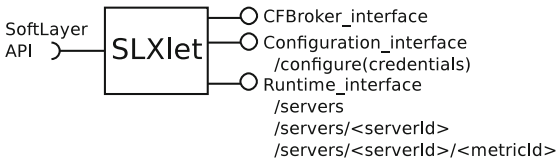

Fig. 3. Xlet interface design

Using a PaaS as Xlet platform, in our case the Bluemix implementation of Cloud Foundry, has advantageous characteristics: *scalability* is inherited from the scalability of the Bluemix environment. Since the Xlet can be provisioned as an application or a service within Bluemix, it is easy to scale it horizontally to cope with the work load by adding or removing new instances. All Xlets have a common and generic core code that often allow the easy *reusability* with minor modifications for specific use cases. *Managing* Xlets is handled to Bluemix, the management here includes provisioning, deprovisioning, binding and unbinding Xlets to the rSLA service. Xlets can be integrated *flexibly* using Bluemix services and can be provisioned using different plans, e.g., shared or dedicated. The rSLA architecture itself does not depend on a specific PaaS code base or service.

5.3 Case Study

We have conducted a pilot of the rSLA service to support the management of Cloud services used by an existing customer. In one agreement, the customer migrated a workload from a customer-owned, on-premise data center environment to the IBM Softlayer Infrastructure-as-a-Service. Along with the move, the customer required the monitoring of seven custom SLOs that had never been offered previously by the service provider in the Cloud in this form.

Each of the seven SLOs consist of one base metric and one service level objective, with multiple composite metrics used to aggregate to level needed by the SLOs. The rSLA document has been defined based on the written agreement with the customer and discussions with the client. The rSLA was submitted to and executed by the rSLA service to activate and initiate the measurement of the involved base metrics. The rSLA service monitors, measures, evaluates and reports the service level status of the seven involved SLOs on a daily basis, resulting in about 100 MB/month in observation data.

We developed three different Xlets to monitor various aspects of the service such as network status and provisioning process times, as well as one Xlet for email notification. Overall, it took 3 weeks from first contact with the project team to the start of monitoring. This included the development of the specific Xlets, the definition of the SLA document, and obtaining the access keys to the client's Cloud service API. This is much faster than a traditional integration to an enterprise SLA management system. In a future scenario, these Xlets could be reused for another client of Softlayer requesting an SLA related to the same base metrics. The remaining effort only comprises the writing of the SLA document and obtaining the API keys for monitoring Xlet configuration, which may be as little as a number of hours or a day. Over time, having acrued a variety of Xlets for various service interfaces, SLA monitoring for popular services can be set up just based on an rSLA document.

6 Summary and Conclusions

Today's cloud-based application environments enable their users to deploy and change applications constantly, binding to different services and platforms on

short notice. Traditional enterprise SLA management typically requires a complicated setup process lagging far behind the application life cycles of a DevOps environment. Current industry practice to Cloud SLA management often fails to take into account specific customer needs. Existing approaches for Web and Grid services often fail to deal with interface heterogeneity in an effective way and have a syntax that is often perceived to be cumbersome by practitioners.

The rSLA approach presented in this paper addresses the efficient specification of SLAs in a formal language and, at the same time, uses the Xlet architecture abstractions to overcome issues of heterogeneity. While reusing some existing concepts such as metrics and SLOs, rSLA makes a number of significant contributions to meet the objective of fast SLA deployment in a Cloud environment: Xlets provide a standard way to refer to and use diverse interfaces. The concept of rSLA measurement directives ties base metrics to the way metrics can be obtained through Xlets, thereby enabling references to metrics from various interfaces in the language. The design of the rSLA language as a Ruby DSL provides access to a full expression language in a way that is familiar to many target users such as administrators using Chef.

The approach has been implemented on Bluemix and tried out in a pilot, monitoring IaaS-related SLOs. This has been accomplished much faster than using an enterprise SLA management system and we were able to deal with a previously unknown management API, which Web service-oriented systems such as WSLA cannot. In addition, the resulting SLAs are actually legible. While we have obtained first results of our approach presented in this paper further work will be required addressing expressiveness of different scenarios, performance, and user acceptance.

The current implementation still has some limitations: it does not allow a user to specify his own measurement mechanisms and xlets are bound statically. We expect to address some of those issues in future work.

References

1. Andrieux, A., Czajkowski, K., Dan, A., Keahey, K., Ludwig, H., Nakata, T., Pruyne, J., Rofrano, J., Tuecke, S., Xu, M.: Web Services Agreement Specification (WS-Agreement), September 2005. http://mailman.ogf.org/documents/GFD.107.pdf
2. Baset, S.A.: Cloud slas: present and future. SIGOPS Oper. Syst. Rev. **46**(2), 57–66 (2012)
3. Butler, J., Lambea, J., Nolan, M., Theilmann, W., Torelli, F., Yahyapour, R., Chiasera, A., Pistore, M.: SLAs empowering services in the future internet. In: Domingue, J. (ed.) The Future Internet. LNCS, vol. 6656, pp. 327–338. Springer, Heidelberg (2011)
4. Comuzzi, M., Kotsokalis, C., Spanoudakis, G., Yahyapour, R.: Establishing and monitoring slas in complex service based systems. In: IEEE International Conference on Web Services 2009, ICWS 2009, pp. 783–790, July 2009
5. Dan, A., Davis, D., Kearney, R., Keller, A., King, R., Kuebler, D., Ludwig, H., Polan, M., Spreitzer, M., Youssef, A.: Web services on demand: WSLA-driven automated management. IBM Syst. J. **43**, 136–158 (2004)

6. Cloud Management Working Group, D.: Cloud infrastructure management interface (CIMI) model and RESTful HTTP-based protocol an interface for managing cloud infrastructure, dsp0263. Technical report, Distributed Management Task Force, Inc. (2014)
7. Kouki, Y., de Oliveira, F., Dupont, S., Ledoux, T.: A language support for cloud elasticity management. In: 2014 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), pp. 206–215, May 2014
8. Liu, F., et al.: SP 500-292 Cloud Computing Reference Architecture, September 2011
9. Ludwig, H., Dan, A., Kearney, R.: Cremona: an architecture and library for creation and monitoring of WS-agreements. In: Proceedings of the 2nd International Conference on Service Oriented Computing (ICSOC 2004), pp. 65–74. ACM (2004)
10. Ludwig, H., Keller, A., Dan, A., King, R., Franck, R.: Web Service Level Agreement (WSLA) Language Specification. Technical report, IBM Corporation, January 2003
11. Ludwig, H.: rSLA language specification. Technical report, IBM research Almaden USA, May 2015
12. Mettraux, J.: rufus-scheduler (2005). https://github.com/jmettraux/rufus-scheduler. Accessed January 2015
13. Michlmayr, A., Rosenberg, F., Leitner, P., Dustdar, S.: Comprehensive QoS monitoring of web services and event-based SLA violation detection. In: Proceedings of the 4th International Workshop on Middleware for Service Oriented Computing, MWSOC 2009, pp. 1–6. ACM, New York (2009)
14. Muller, C., Oriol, M., Franch, X., Marco, J., Resinas, M., Ruiz-Cortes, A., Rodriguez, M.: Comprehensive explanation of SLA violations at runtime. IEEE Trans. Serv. Comput. **7**(2), 168–183 (2014)
15. OCCI: Open Cloud Computing Interface (2010). http://occi-wg.org/
16. Torkashvan, M., Haghighi, H.: Cslam: a framework for cloud service level agreement management based on WSLA. In: 2012 Sixth International Symposium on Telecommunications (IST), pp. 577–585, November 2012
17. Tosic, V., Patel, K., Pagurek, B.: WSOL - web service offerings language. In: Bussler, C.J., McIlraith, S.A., Orlowska, M.E., Pernici, B., Yang, J. (eds.) CAiSE 2002 and WES 2002. LNCS, vol. 2512, pp. 57–67. Springer, Heidelberg (2002)
18. Zeng, L., Benatallah, B., Dumas, M., Kalagnanam, J., Sheng, Q.Z.: Quality driven web services composition. In: Proceedings of the 12th International Conference on World Wide Web, WWW 2003, pp. 411–421. ACM, New York (2003)