Table of Contents

1 rSLA DSL Introduction

How can rSLA become the dominant language in the market (not standard).

This document describes the structure, vocabulary and grammar (production rules) of the rSLA DSL. rSLA stands for ruby SLA and is implemented as a domain specific language (DSL) for the definition and management of service level agreements (SLAs) for cloud service provisioning. The language consists of the rSLA runtime that processes rSLA instances in a cloud environment like IBM Bluemix, of the rSLA editing environment and of Java xLets for monitoring and scheduling existing as well as new rSLA instances.

rSLA accommodates the WSLA specification [**?**] in the cloud setting. rSLA provides a concrete computing framework for the definition and management of cloud SLAs. The DSL uses Ruby code blocks, i.e. anonymous functions, to express SLA data and to describe their operations and functionality in an SLA runtime environment. Ruby blocks are perceived as closures that allow for customized definition of SLA properties, for their nesting with other rSLA components and their flexible handling in the program flow.

The following sections specify the semantic significance of each object in the rSLA language, their logical definition, relationships to other objects and production rules for SLA formation and management.

2 WSLA motivation

rSLA is motivated by the WSLA specification [**?**], whose data structure is depicted in Figure **??**. The WSLA specification defined the SLA structure for web services that are provisioned over distributed computing settings. The specification was originally implemented with the Java programming language and signified the state-of-the-art for numerous research efforts in the grid and utility computing domains. WSLA used the XML standard [**?**] to define and express SLA language concepts and grammar. First order logic was used to describe production rules and relational dependencies of inclusive elements.

The DSL inherits language concepts from WSLA. Figure **??** illustrates primary components of the rSLA language and their cardinality with respect to an SLA instance.

According to the WSLA specification [**?**], an SLA consists of three primary sections:

1. parties that specify the two signatory parties (customer - provider) and any support parties for the service execution,
2. service description or definition that specifies the agreed service context in terms of defined SLA parameters and metrics,
3. obligations or service level guarantees that define agreed responsibilities and promises with respect to service level values for all involved parties.

The pilot version of the rSLA language is currently deployed as a ruby sinatra web service and tested on the IBM Bluemix cloud platform.

3 rSLA BNF

A BNF language helps to formally describe the syntax of a programming language. The text describes grammars for the rSLA DSL using Backus-Naur form notation. An rSLA grammar can be extended to run as CFG (context-free grammar), which is helpful with writing new programming language rules.

In the rSLA BNF definition, every grammar rule is decomposed into another set of rules and/or literals. Non-terminal symbols are denoted as $< non-terminal-symbol >$ and the ::= symbol is translated as "expanded to" or "replaced with". In the text, terminal symbols or literals are denoted as 'literal'.

Listing 1: rSLA BNF grammar rules

```
1 grammarRule $::= <rule>+ 'literal' | <rule>$ ;
2 "|" ::= choice, logical OR
3 ";" ::= end of rule
4 "::=" ::= defined as/consists of
5 "?" ::= suffix to denote optional
6 "()" ::= denote grouping of rules and/or literals
7 Kleene star * for 0 or more
8 Kleene plus + for 1 or more
9 ".." ::= range
```

```
<digit> ::= '0'..'9'
<dow>   ::= 'a'..'z'
<number>  ::= <digit+> | <dow+>
<Id> ::= 'literal'| ('a'..'z'|'A'..'Z'|_|'0'..'9')* ;
<dateTime> ::= see ISO 8601 date time format
<string> ::= """<Unicode content>"""
<Unicode content> ::= See http://www.unicode.org/versions/
    Unicode7.0.0/
<URI> ::= See RFC 3986 (http://www.ietf.org/rfc/rfc3986.txt)
```

3.1 rSLA vocabulary

The rSLA language inherits from the WSLA specification [**?**] the semantic decomposition of an SLA into a hierarchical tree, where the root node represents an SLA object. Nodes attached to the root denote SLA branches (e.g. base and composite metrics, service level objectives). Figure 1 illustrates a class diagram

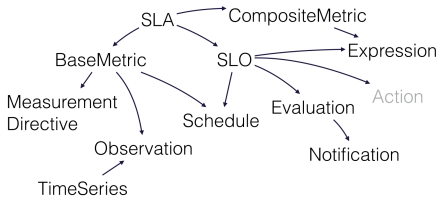

Fig. 1: rSLA DSL user input diagram

of the core rSLA DSL alphabet. Solid black arrows in Figure 1 indicate that user-input is required to create the equivalent objects in an rSLA running environment. The alphabet is illustrated as a tree, where connections between nodes highlight relationships and nesting of SLA context.

The rSLA alphabet consists of language elements that require user input for their creation in an rSLA runtime environment. A DSL user can directly edit code-block scripts in ruby, start a cloud rSLA service and create an active SLA object.

In Figure 1 language elements that require user-input represent tree branches for the creation of service level agreements. DSL user-input requires editing a ruby script to describe the attributes for any new rSLA object. The structure of such script is discussed in the following paragraphs as it is intepreted by the rSLA engine.

The rSLA language also contains elements, whose definition with an rSLA service requires that at least one SLA object exists (ex. base metric, slo). A DSL user can edit the context of such elements and associate them with required objects using code blocks in ruby SLA scripts.

3.2 SLA object

rSLA enables the generation and processing of SLA instances in a cloud runtime environment, where an rSLA service is deployed and running. The grammar excerpt in BNF notation illustrates the rSLA language rules for producing SLA objects:

Listing 2: $< SLA >$ BNF grammar

```
1  <SLA> ::= (<baseMetric+>) (<compositeMetric*>)(<SLO+>) <
       customer> <provider> <description> <agreedTime>;
2 <description> ::= 'literal';
3 <customer> ::= 'literal';
4 <provider> ::= 'literal';
5 <agreedTime> ::= 'date-time literal';
```

rSLA exposes SLA information through ruby programming blocks that are simple to edit and process. The rSLA DSL provides an SLA class for the generation of SLA objects that conform to attributes of ruby blocks as the following:

Listing 3: rSLA SLA definition

```
1 sla do
2         tenant "XYZ Inc."
3         provider "IBM"
4         description "This is a sample SLA"
5         creationTime "2015-01-30"
6 end
```

An SLA object is defined by a description, an id and contains a date identifier to timestamp an object's creation time. A generated SLA is related to a single provider and a single customer, which are referenced by their unique ids. Additionally, an SLA is associated with:

- 1 or more baseMetrics (section 3.3),
- 0 or more compositeMetrics (section 3.5) and
- 1 or more SLOs (section 3.6).

The illustrated block provides information to the rSLA runtime environment regarding contracting entities, i.e. service customer and provider. rSLA defines a single customer and a single provider rule for every SLA object [**?**]. A DSL user can set the identity of respective parties through such properties and also extend them to reflect domain specific needs.

Every new SLA object is identified by a unique id. The SLA identifier is used by the rSLA runtime to index an SLA object into available persistence middlewares and to associate it with related objects in the runtime environment.

3.3 Base metric

A base metric inherits the semantics of resource metrics by the WSLA specification [**?**] and adjusts their meaning into the cloud setting. The rSLA language implements a simple BNF grammar for the definition of *baseMetric* objects:

Listing 4: $< baseMetric >$ BNF grammar

```
<baseMetric> ::=  <name> <type> <source> <unit> <
    measurementDirective> <schedule+> ;
<name> ::= 'literal';<string>
<type>
<source>
<unit>
<measurementDirective> ::= <measurementDirective>;
<schedule> ::= <schedule>;
```

The rSLA DSL implements the $< baseMetric >$ grammar into a class that generates *baseMetric* objects in an rSLA runtime environment. A base metric represents a service metric that is used by one or more SLAs for the assembly of composite metrics 3.5 or for service level evaluation at runtime.

The values of base metrics are monitored and collected according to specified schedules. Such values are either persisted or directly processed to derive new values that are used for service level assessment.

A base metric is defined by a unique id and by a name. A base metric can also be described by a type that designates the metric data structure, by a measurement unit and by a source attribute to the URI[1] endpoint for the monitoring and the observing of base metric values.

Listing 5: rSLA baseMetric definition

```
basemetric do
    name "CPUmetric1"; type "float"; source "nxlet"; unit "Hz"
    measurementdirective do
                entity "public"
                type "jsonArray"
                source "http://nxlet.mybluemix.net/usage"
        end
        schedule do
        frequency "1"
        unit "m"
        method "every"
        end
 end
```

Every base metric has a mapping to a single measurement directive (section 3.3), which in rSLA represents a data structure that provides directives on the monitoring and value-collection of each generated base metric.

A base metric is also mapped to a schedule (section 3.4) that specifies the monitoring and measurement periods for the base metric values. Similar to measurement directives, the mapping between base metrics and schedules is currently one-to-one.

A base metric references its parent SLA object by the SLA object id. A base metric may belong to one or more SLA objects? Additionally, the attributes of a

[1] Unique Resource Identifier, `http://en.wikipedia.org/wiki/Uniform_resource_identifier`

base metric object can be used for the aggregation or composition of composite metrics (section 3.5) and for the assessment of service level objectives (section 3.6).

Measurement directive In the rSLA DSL a measurement directive represents a data structure that indicates how the values of base metrics can be monitored and fetched for further processing. The semantic notion is inherited from the WSLA specification [**?**].

Listing 6: $< MeasurementDirective >$ BNF grammar

```
1 <measurementDirective> ::=  <entity> <type> <source>;
2 <entity>, <type>, <source>  ::=  'literal';
```

According to WSLA, a measurement directive describes how to read a metric value from instrumentation. For rSLA, such instrumentation is provided by Java Xlets (Section **??**runtime) that are deployed as Java liberty services in the rSLA runtime environment.

In WSLA, an entity refers to a "thing" being measured, e.g. a process, a server, a network. In rSLA language, the term $< entity >$ is used for the representation of restful [2] endpoints. Such may have a URL[3] form: `http://a.cloud.com/servers/s3456/metrics/cpuconsumption`. Here the restful endpoint specifies a metric with respect to CPU consumption.

The rSLA language provides a measurement directive class for the generation and customization of measurement directive objects, which in turn are referenced by *baseMetric* ones . An rSLA ruby block for the definition and editing of measurement directives is illustrated below:

Listing 7: rSLA measurementDirective definition

```
1 measurementdirective do
2         entity "replacement"
3         type "jsonArray"
4         source "http://hruxlet.mybluemix.net/hardware"
5 end
```

On definition, a measurement directive object is described by a unique id, an entity, a result type and a source URI. A measurement directive indicates the result type that is expected from a base metric measurement. In the above snippet the result is expected as an array of json objects, whereas in other cases the result type may indicate the data type of the measurement output.

Moreover, a measurement directive object uses an attribute named *source* to denote the restful endpoint for fetching the relevant base metric value. The rSLA $MeasurementDirective$ class provides an example on how to define measurement directive objects for rSLA base metrics and can be extended accordingly.

[2] ReST: `http://en.wikipedia.org/wiki/Representational_state_transfer`

[3] Unique Resource Locator: `http://en.wikipedia.org/wiki/Uniform_resource_locator`

3.4 Schedule

Information defined in an SLA is monitored systematically as it helps to assess and verify adherence on agreed service levels. Updates of service level values, like those of base metric ones, are performed methodically and synchronized with the processing of other dependent rSLA object values. In SLA management [**?**] a "dependent" entity requires the values of other entities to define its own.

For such reasons a scheduler is required to coordinate SLA operations. Currently, the rSLA runtime can integrate a variety of ruby job queuing systems, like clockwork[4] or some ruby flavor of a more traditional Cron [5] scheduler. The schedule context represents fundamental information for a scheduler to operate orderly. The BNF grammar listing illustrates simple rules to produce such scheduler context:

Listing 8: < *Schedule* > BNF grammar

```
1 <schedule> ::= <frequency> <unit> <method> ;
2 <frequency> ::=  'date-time literal' ;
3 <unit> ::=  'literal' ;
4 <method> ::=  <compoundStatement> | <compoundStatement> + '
    literal' | 'literal' ;
5 <compoundStatement> ::= <compoundStatement> + <statement> | <
    statement> | <compoundStatement> ;
6 <statement> ::= <statement>+<term> | <statement> | <term> ;
```

A schedule can be decomposed into its operation frequency, unit and triggering method. The operation frequency and unit provide information as to how often a scheduled task is running. The triggering method tells the scheduler how to execute the task. In rSLA a schedule takes the form of a nested array that passes such schedule parameters to a backend message queuing system. The DSL exposes a ruby block for a schedule definition and configuration that is encapsulated in the listing 9

Listing 9: rSLA schedule definition

```
1 schedule do
2         frequency "1"
3     unit "m"
4     method "every"
5 end
```

The pilot version of the rSLA runtime uses the rufus-scheduler[6] to facilitate job coordination. Rufus requires the input of three attributes to operate. Such attributes represent the schedule periodicity or frequency along with the schedule time unit, e.g. '1m' for one minute or '2h' for 2 hours.

Rufus also awaits for a method to trigger the scheduled job. For example a scheduled task can be fired every minute or every other day. Additionally, for

[4] `https://github.com/tomykaira/clockwork`
[5] Cron: `http://en.wikipedia.org/wiki/Cron`
[6] `https://github.com/jmettraux/rufus-scheduler`

one time and non-frequent jobs the "at" and "in" clauses can be used for the schedule configuration. Combinations between such clauses are possible.

3.5 Composite metric

A composite metric represents a service metric that is composed by the values of other base and/or composite metrics [**?**]. WSLA introduced this term to denote metrics that are defined in one or more SLAs and are composed by one or more base and/or composite metric values. In the rSLA DSL, a composite metric is represented by the BNF grammar in listing 10:

Listing 10: $< CompositeMetric >$ BNF grammar

```
1 <compositeMetric> ::= <name> <type> <unit> <expression>;
2 <name>, <type>, <unit> ::= 'literal' ;
3 <expression> ::= <metrics+> <method> <select> | <
      compoundStatement> + <statement> | <statement> ;
4 <metrics> ::= <baseMetric*> <compositeMetric*> ;
5 <method> ::= <compoundStatement> | <compoundStatement> + '
      literal' | 'literal' ;
6 <compoundStatement> ::= <compoundStatement> + <statement> | <
      statement> | <compoundStatement> ;
7 <statement> ::= <statement>+<term> | <statement> | <term> ;
8 <select> ::= 'date-time literal' ;
```

The rSLA DSL provides a *CompositeMetric* class to represent grammar 10. The rSLA language exposes ruby code blocks, like the ones in listing 11, for the representation of*CompositeMetric* objects in an rSLA runtime environment.

Listing 11: rSLA composite metric definition

```
1 compositemetric do
2     name "CPUutilization"
3     type "float"
4     unit "Hz"
5     expression do
6                 (CPUmetric1+CPUmetric2)/2
7         end
8         schedule do
9         frequency "15"
10        unit "m"
11        method "every"
12        end
```

A composite metric object is assigned a unique id and contains name, type and unit-measurement attributes that can be used to create customized index criteria. The value of a composite metric represents the result of functions. In the rSLA language the notion of function is enclosed within a ruby programming block that is called expression. The Expression class (section 3.7) provides a reference for the creation of valid expression objects.

For rSLA composite metric objects an expression represents a formula or function for their value computation. Such formula may refer to values of other base and/or composite metrics. An rSLA expression can be defined by any set of valid ruby statements that can be expressed and produced by an rSLA grammar.

3.6 Service level objectives

Service level objectives (SLOs) illustrate promises from a provider to a customer with respect to service provisioning levels[**?**]. An SLA is defined by one or more SLOs. The rSLA DSL uses the BNF grammar of listing 12 to summarize production rules for the generation of an SLO object:

Listing 12: $< ServiceLevelObjective >$ BNF grammar

```
1 <SLO> ::= <name> <precondition> <objective> <schedule> ;
2 <name>::= 'literal' ;
3 <precondition>::= <expression> ;
4 <objective> ::= <expression> ;
5 <compoundStatement> ::= <compoundStatement> + <statement> | <
      statement> | <compoundStatement> ;
6 <statement> ::= <statement> + <term> | <statement> | <term> ;
7 <schedule> ::= <schedule> ;
```

The rSLA language provides an *SLO* class for the generation of SLO objects that follow the rules of grammar 12. An SLO object has a name and an evaluation schedule. An SLO also defines a precondition statement block that is coupled to an objective statement block. The logical relationship between a precondition and an objective block is hierarchical and summarized as follows:

if $eval(Precondition) \rightarrow \neg Precondition$ **then**
  $SLO_{healthy}$ = true
else if $\nexists Precondition \wedge eval(Objective) \rightarrow$ true **then**
  $SLO_{healthy}$=true
else
  $SLO_{healthy}$=false
end if

An rSLA SLO object will first evaluate the precondition statement block. If the logical outcome from the execution of the precondition block is false, the SLO is healthy. In case the precondition is true (or if there is no precondition block) the objective block is evaluated. If the logical outcome from running the objective block is true, the SLO is healthy, otherwise the SLO evaluation indicates not-healthy. A non-healthy SLO may result into a violation.

The rSLA DSL exposes ruby programming blocks for the production of SLO objects. A DSL user can edit and extend such programming blocks according to domain specific needs. An example of an rSLA SLO programming block is illustrated at listing 13.

Listing 13: rSLA SLO definition

```
slo do
     name "CpuUtil"
     precondition do CPUutilization.value<15 end
     objective do CPUmetric1.value<10 end
     schedule do
        frequency "30"
        unit "m"
        method "every"
    end
end
```

The SLO definition in listing 13 provides a configuration sample for the generation of SLO objects with the rSLA language. In listing 13 the precondition and objective couple are defined using simple statements, which might not be the case with a real rSLA runtime scenario.

3.7 Expression

In rSLA language an expression can be generated using the BNF production rules of listing 14. Such production rules and generated grammar can be further customized and extended to reflect specific domain needs:

Listing 14: < *Expression* > BNF grammar

```
<expression> ::= <term> | <compoundStatement> + <term>  | <
    compoundStatement> ;
<compoundStatement> ::= <compoundStatement> | <
    compoundStatement> + <statement> | <statement>;
%here it becomes problematic
<statement> ::= <statement> | <statement>+<term> | <term> ;
<term> ::= <term> + 'literal' | <term> | 'literal' ;
```

rSLA supports the ruby flexibility in programming with free form expressions. Under free form expressions the rSLA DSL encloses the free-of-syntax definition of rSLA object relationships that may specify a set of condition statements with respect to the logical comparison of such rSLA object values.

The free of syntax definition of conditional relationships for the evaluation and processing of service values listens to the rSLA language production rules. Such rules can be extended to the Ruby programming language perimeters. Expressions represent a core element for the description of composite metric, SLO objects and other rSLA language objects.

4 rSLA BNF (user implicit?)

4.1 Timeseries

A time-series is defined by a sequence of data points that are typically collected from successive observations[7]. rSLA defines a sample grammar in BNF notation to describe a time-series object in an rSLA environment:

[7] Time-series: `http://en.wikipedia.org/wiki/Time_series`

Listing 15: < *timeSeries* > BNF grammar

```
1 <timeseries> :: = <start-point>..<end-point+> | <start-point
      +>..<end-point> | <start-point>..\infty ;
2 <start-point> :: =  'date-time' + <compoundStatement> | 'date-
      time' .. \infty | 'date-time' ;
3 <end-point> :: = <start-point> + <compoundStatement> + <digit>
       | <start-point> + <digit> | <digit> ;
4 <digit> ::= "0" | <positive_digit>
5 <positive_digit> ::= "1" | "2" | "3" | "4" | "5" | "6" | "7" |
       "8" | "9" ;
```

The rSLA language provides a TimeSeries class to represent the BNF grammar of listing 15. rSLA time-series objects help with the evaluation of SLOs Such evaluation can be schedule specific (eg. every-month) or may require the application of time-series operations. In the latter case, a set of statistical or other numerical functions apply on a series of observations that are needed for the SLO evaluation. Such evaluation functions use time-series objects to process the expect evaluation output.

4.2 Observation

Grammar

```
Observation -> id value observeTime extras month baseMetricId
```

4.3 Evaluation

Grammar

```
Evaluation -> id SLOid value timestamp
```

4.4 Notification

Grammar:

```
Notification -> id SLOid reportPeriod context createdTime
```

4.5 Action

An action designates operations to be taken in case of events like violations or notifications during an SLA runtime. The notion is inherited from WSLA, where Actions represented activities performed by parties during an SLA runtime. Such activities were triggered by party instances. In the rSLA DSL, actions are triggered by events during an SLA runtime. rSLA defines Action objects (Section 4.5) that are referenced by SLOs and invoked in case of SLO violations. Actions may trigger alerts as well as corrective operations for entities that are involved with an SLO violation or relevant event. Precondition and objective blocks are determined by one or more SLA intervals and thus may change within the total lifetime of an SLA.

5 rSLA runtime environment

5.1 xLets

what is the xLets role in the rSLA environment: what is an xlet, what is its functionality, which xlets rSLA has now, easy to implement/deploy new xlets (connection link) for rSLA runtime environment

5.2 data management

what exists now, give example: how calculate metrics, how evaluation is done, how notifications are created and sent

5.3 IBM Bluemix deployment

summary on how the rSLA service can be deployed on a Bluemix environment (kid of descriptive)