Describing a monitoring infrastructure with an OCCI-compliant schema

Augusto Ciuffoletti - Dept. of Computer Science - Univ. of Pisa

December 19, 2012

Abstract

The OCCI Extension for the resource monitoring infrastructure describes the API needed to inspect the operation of functional resources. The definition entails the introduction of two further *Kinds*: the *Sensor Resource*, that processes metrics, and the *Collector Link*, that extracts and transports metrics. They are defined as generic *Kinds*, that are specialized using OCCI *Mix-in*s. Using this API, the user isprovided *on demand* with an amonitoring infrastructure.

The provision of this API is considered as an option for the provider.

One relevant target of this document is the design of an API for SLA: under this light, the API for the Resource Monitoring Infrastructure offers the tools to verify and implement the Service Level Objectives (SLO).

The paper is divided into three parts: an introduction (20%) that gives the feeling of the schema, a core part (50%) that defines the API, and finally an appendix with an example, with reference to an HTTP rendering.

1 Introduction

This document describes the structure of a monitoring infrastructure using the OCCI restful model. It is intended to be a first step towards the definition of a protocol to manage and verify Service Level Agreement (SLA), but it can extend its application also outside SLA.

The purpose of a specification for the management interface of a monitoring infrastructure is that of giving the possibility to arrange it in a way that suits user's needs, instead of being limited to the SLA provided by the server. The existence of a standard specification makes it possible for the user to manage distinct cloud providers, possibly at the same time, using the same interface.

The importance of a configurable monitoring infrastructure emerges in complex scenarios, where the user is in fact an intermediate service provider, that provides SLA services to third party users: in that case, the intermediate provider may decide to provide SLA options that differ from that of the low level provider. In that case the intermediate provider may want to perform measurements on the infrastructure leased by the low level provider(s).

The management capabilities should also extend to the adaptive, and dynamic configuration of the components that contribute to the monitoring: the specification schema must allow the user to explore the available functionalities in order to adaptively arrange a monitoring infrastructure, and to modify them according with changing needs.

One relevant fact about monitoring infrastructures is that it is extremely difficult to give a *detailed* framework for them that extends its validity to any reasonable use case or provider. The reason is that each use case and provider exhibits local variants that do not fit a rigid *standard* approach. Also, the metrics that are used to evaluate the performance of the system are many, and subject to continuous changes due to the introduction of new technologies. Thus we have made an effort to introduce a generic schema that can be adapted to effectively describe the relevant aspects of a monitoring infrastructure, but that does not interfere with details that depend on the specific environment.

The OCCI Core Model [3] is well suited for the task, since it embeds the tools needed to extend a framework with provider specific details: this enables the specification of the abstract

model, leaving to the user the task of making explicit the details, targeting a specific provider or technology. Furthermore, we claim that the specifications given in this document can find an application in environments other than computing infrastructures, since we abstract from the details that characterize cloud infrastructure resources.

The approach followed in this document is similar to that found in the infrastructure document (GFD-P-R.184 [4]): the monitoring capability is associated with a new *Kind*, the *Sensor*, that is related with the OCCI Core Model Resource type. A *Sensor Resource* instance is a black box that collects metrics from its input side, and delivers aggregated metrics from its output. The input and output channels are modeled with another *Kind*, called *Collector*, that is related with the OCCI Core Model Link type. The role of a *Collector Link* instance is twofold: on one side it indicates the way metrics are conveyed to the target *Resource*, on the other it indicates a specific monitoring technique applied on the source *Resource*. Both capabilities are controlled with *Mix-in*association. For instance, the provider may offer the client a specific *Ping Mixin* that can be associated with a *Collector Link* originating from a *Network Resource*: such a configuration will deliver metrics like the roundtrip time, and the packet loss rate to the attached *Sensor Resource*. To enable the discovery of such *Mix-in*s, they are gathered under specific collections.

Using these basic building blocks the designer is able to assemble complex, multilayer monitoring infrastructures: for instance, a *Sensor Resource* can be used to aggregate a storage throughput using the input from three *Collector Link*s, one for the average response time, one for the mean time between failures, and another for network delay, and provide the results to an upstream *Sensor Resource* that aggregates the same results from other *Sensor Resource*s. On the other hand, the model is able to describe very simple scenarios, like a compute *Resource*that logs its activity in a database hosted by a storage *Resource*: in that case one *Collector Link* connects the compute resource to the storage, without the need of a Sensor. Appropriate *Mix-in*s associated with the *Collector Link* describe the logging activity and the database access mode.
[1]

Note that the schema is transparent, in particular, to the existence of a standard for metric identifiers: if one exists, the interoperability of distinct monitoring infrastructures is certainly improved. We consider that the user that interacts with the monitoring infrastructures either knows about the identifiers used by the provider, or uses an interface (e.g., a SLA negotiation service) that translates provider specific identifiers into interoperable ones. This document highlights further standardization issues.

Summarizing, the specification introduced in this document requires that the conformant provider implements two *Kind*s: the *Sensor Resource* and the *Collector Link*. Three tagging *Mix-in*s are also defined, namely *ToolSet*, *CollectorSet* and *AggregatorSet*, to identify the collection of mixins that describe the specific capabilities associated with an instance of the above *Kind*s.

1.1 Terminology shortcuts

We will use the term <*mixin id*> *collection* to indicate the set of *Mix-in*s that are associated with the identified tagging *Mix-in*. The provider ensures that the *Mix-in*s in a given collection have defined semantics, as explained in the rest of this paper.

To distinguish a *Resource* instance from its *Kind*, we will use the indeterminative article for the instance (e.g., "a *Resource*"), and the determinative article for the *Kind* (e.g., "the *Resource*"). The plural is reserved to instances (e.g., "the *Resource*s"). In case of ambiguity we will further specify "instance" or "*Kind*".

[1] **Remark by author**: Why not a mixin to the monitored resource directly? I envision problems emerging with the implementation. A resource can be "prepared" for monitoring, but the way in which the Monitoring Link will interact with such preparation is not clear. In addition, consider that the same tool might be the target of several links, with distinct configuration parameter. How can the control parameters of the mixin be exposed in such a case? Instead, if the mixin is embedded in the link, it is the responsibility of the link implementation to configure it, and to couple it with the publishing technology indicated in the link

Model attribute	value
scheme	http://ogf.schemas.sla/occi/monitoring#
term	sensor_tpl
attributes	None

Model attribute	value
scheme	http://ogf.schemas.sla/occi/monitoring#
term	tool_tpl
attributes	None

Model attribute	value
scheme	http://ogf.schemas.sla/occi/monitoring#
term	publish_tpl
attributes	None

Table 1: Definition of the template *Mix-in*s

2 Specification of the compliant server

The compliant server MUST define the following *Kind*s:

Sensor Resource that describes how monitoring results are aggregated (see table 2);

Collector Link that describes how monitoring results are trasferred between *Resource*s (see table 3);

In addition, the compliant server MUST define the following *Mixin*s (see table 1):

AggregatorSet that is used to tag the *Mix-in*s describing the aggregation function operated by a *Sensor Resource*;

ToolSet that is used to tag *Mix-in*s describing Monitoring tools associated with a *Collector Link*;

CollectorSet that is used to tag the *Mix-in*s that describe the technique used to transport monitoring results in a *Collector Link*;

2.1 Constraints on instances

The constraints defined on the instances of the *Kind*s and *Mixin*s defined in the previous section are as follows:

- a *Sensor Resource* MUST be the *target* of at least one *Collector Link* and MUST be the *source* of exactly one *Collector Link*;
- a *Mix-in*in the [ToolSet] collection can be associated ONLY with a *Collector Link*;
- a *Mix-in*in the [CollectorSet] collection can be associated ONLY with a *Collector Link*.
- a *Mix-in*in the [AggregatorSet] collection can be associated ONLY with a *Sensor Link*

[2]

2.2 The *Sensor Resource*

The *Sensor Resource* models the manager of a monitoring activity that encompasses the collection of measurements, their aggregation in composite metrics, and their delivery to the user.

A *Sensor Resource* is characterized by attributes that define the rate with which new observations are produced, and by the scheduling times of its operation (see table 2). The attributes with the **req** label MUST be assigned a legal value upon instantiation. The server MUST reject an incomplete instantiation.

The execution rate is defined using three attributes: the rate itself, and an optional definition of the quality of the timing. This latter attribute contains a triple of numbers encoded as a string, that define the granularity with which the rate is measured, and the accuracy

[2]**Remark by author**: The utilization of *Mix-in*s, instead of kind-specific attributes describing the operation, has the purpose of allowing the discovery of the capabilities offered by the provider. Kind specific attr. might be three, describing the tool id, and ohter two formatted strings for the input and output parameters

Model attribute	value
scheme	http://ogf.schemas.sla/occi/monitoring#
term	*Sensor Resource*
attributes	(see below)
related	http://ogf.schemas.sla/occi/core#resource

Set of Attributes for the *Sensor Resource*				
name	type	mut.	req.	Description
occi.sensor.period	number	yes	yes	The time between two following measurements
occi.sensor.periodspec	string	yes	no	granularity, accuracy, exponent of period measument
occi.sensor.timebase	number	no	yes	The server time when the timestart and timestop are modified
occi.sensor.timestart	number	yes	yes	The delay after which the session is planned to start
occi.sensor.timestop	number	yes	yes	The delay after which the session is planned to stop
occi.sensor.timespec	string	yes	no	granularity, accuracy, exponent of time measurement

Table 2: Definition of the *Sensor Resource* Kind

of rate measurement, and the floating point exponent. By default `periodspec` is set to `NaN, NaN, 0`.

The activation of a *Sensor Resource* is controlled by two attributes that describe the scheduling of sensor activity: to schedule the execution of a sensor the user modifies the `starttime` with a value indicating how far in the future the instance is going to start its activity. A value of zero corresponds to the immediate start. The server sets the `timebase` attribute corresponding to the reference time of the start time.

All time values are represented as numbers. The `timebase` corresponds to Unix seconds, all timing values use a floating point notation. Also for time values there is a `timespec` attribute analogous to `periodspec`.

2.3 The *Mix-in* in the *AggregatorSet* collection

A *Mix-in* instance in the *AggregatorSet* collection is meant to implement the computation of an aggregated metric starting from raw metrics: it represents the function applied by a *Sensor Resource*. In principle, each provider has a distinct offer of such *Mix-in*s, so here there is ground for further standardization. If the provider does not adhere to a defined standard, it MUST give an exhaustive documentation of the aggregation functions associated with a *Mix-in*.

The attributes of a *Mix-in* in the *AggregatorSet* collection are divided into three groups:

- Input attributes: they bind a metric in the scope of the *Sensor Resource* with an input of the aggregating function. The scope of a *Sensor Resource* consists of the `name`s of all the metric attributes of the incoming *Collector Link*s. A metric indicated as the value of an input attribute MUST be in the scope of the *Sensor Resource*. For instance, a *Sensor Resource* that implements a EWMA may have an `input` attribute equal to

  ```
  data="com.provider.monitoring.collector1.roundtrip"
  ```

  where `roundtrip` is a metric delivered by an incoming *Collector Link* `collector1`.
- Control attributes: they control the operation of the aggregating function (for instance, the gain of an EWMA);
- Metric attributes: they correspond to the metrics delivered through the outgoing *Collector Link*.

To enable interoperabilty, the provider SHOULD follow a defined standard for the naming of input, control and result attributes, but their specification falls outside the scope of this document.

Model attribute	value
scheme	http://ogf.schemas.sla/occi/monitoring#
term	*Collector Link*
source	URI
target	URI
attributes	(see below)
related	http://ogf.schemas.sla/occi/core#link

Set of Attributes for the *Collector Link*				
name	type	mut.	req.	Description
occi.collector.period	number	yes	yes	The time between two following measurements
occi.collector.periodspec	string	yes	no	granularity, accuracy, exponent of period measument

Table 3: Definition of the *Collector Link* Kind

3 The *Collector Link*

The *Collector Link* models the transfer of metric measurements from one *Resource* to another. The tranfer may be motivated, for instance, by the existence of specialized *Resource*s, by administrative reasons, or to cross inter-provider boundaries.

A *Collector Link* is characterized by two aspects: one is the activity that extracts metric measurements from the source *Resource*, and the other is the transport of the measurements to the target *Resource*. Both of them are defined by *Mix-in*s, respectively from the *ToolSet* and from the *CollectorSet* collections.

Regarding the presence of an associated *Mix-in* in the above collections, we distinguish two basic cases:

- if the `target` of the *Collector Link* is a *Sensor Resource*, then *CollectorSet Mix-in*s SHOULD NOT be associated with the *Collector Link* instance, and the provider has enough information to implement an appropriate channel;
- if the `source` of the *Collector Link* is a *Sensor Resource*, then *ToolSet Mix-in*s SHOULD NOT be associated with the *Collector Link* instance, since it is not meaningful to monitor a *Sensor Resource*.

The metric attributes of the *Mix-in*s associated to the *Collector Link* contribute to the scope of the target *Sensor Resource* referenced in sect. 2.2.

3.1 The *Mix-in* in the *ToolSet* collection

The measurement activity is integrated in the *Collector Link* using a *Mix-in* in the *ToolSet* collection. A *Mix-in* in the *ToolSet* collection implements a measurement activity on the *Resource*that is the source of the *Sensor Resource* the *Mix-in* is associated with.

In principle, each provider may associate a different semantic to a given *Mix-in*, so here there is ground for further standardization. If the provider does not adhere to a defined standard, it MUST give an exhaustive documentation of the monitoring tool associated with a *Mix-in*.

Similar to the case of the *Mix-in*s in the *AggregatorSet*, the attributes are divided into two groups:

- Control attributes: they control the operation of the measurement activity. For instance a *Mix-in* implementing a ping tool may have a control attribute defined as

  `name=size,type=string,mutable=1,required=0,default=84`

  The role of the attributes is part of the specification of the specific *Mix-in*.
- Metric attributes: they correspond to the metrics delivered to the target *Sensor Resource*, and SHOULD hold a reasonably updated value for those metrics. In principle, each provider may associate a different semantic to a given *Mix-in*, so here there is ground for further standardization. If the provider does not adhere to a defined standard, it MUST give an exhaustive documentation of the monitoring tool associated with a *Mix-in*.

3.2 The *Mix-in* in the *CollectorSet* collection

How data are delivered is defined by a *Mix-in* in the *CollectorSet* collection.

In principle, each provider may associate a different semantic to similar *Mix-in*s, so here there is ground for further standardization. If the provider does not adhere to a defined standard, it MUST give an exhaustive documentation of the publishing mode associated with this *Mix-in*.

Examples of measurement delivery modes are through a Unix pipe, on demand through a TCP connection, pushed using UDP datagrams, persistently recorded in a database.

The attributes of a *Mix-in* in the *CollectorSet* are divided into two types:

- Input attributes: their value MUST correspond to the name of one of the output parameters of the source *Sensor Resource*.
- Control attributes alter the process used to publish input parameters;

To enable interoperability, the provider SHOULD follow a defined standard for the naming of input and control attributes, but their specification falls outside the scope of this document.

4 Conformance levels

Since the management of a monitoring infrastructure is an optional capability of a cloud service provision, the definition of conformance levels is appropriate.

Compatibility 0 The *Collector Link* and *Sensor Resource Kind* collections MUST NOT be implemented: attempt of instantiating such *Kinds* fails. In an HTTP rendering a POST and GET over these *Resource* collections returns `404 Notfound`. The *AggregatorSet*, *ToolSet*, and *CollectorSet Mix-in* collections MUST NOT be implemented: discovery fails. In an HTTP rendering a GET over the mixins returns `404 Notfound`;

Compatibility 1 The *Collector Link* and *Sensor Resource Kind* collections MUST be implemented, and the user MUST be allowed to create new instances of such *Kinds*. In an HTTP rendering a POST and GET over these *Resource* collections returns respectively `201` and `200`. In case of error, the server MUST NOT return`404 Notfound`. The *AggregatorSet*, *ToolSet*, and *CollectorSet Mix-in* collections MUST be implemented, and discovery is successful. The server does not allow the instantiation of new Mixins in the *AggregatorSet*, *ToolSet*, and *CollectorSet* collections. In an HTTP rendering, a POST over these mixins returns `405 Method Not allowed`;

Compatibility 2 The *Collector Link* and *Sensor Resource Kind* collections MUST be implemented, and the user MUST be allowed to create new instances of such *Kinds*. In an HTTP rendering a POST and GET over these *Resource* collections returns respectively `201` and `200`. In case of error, the server MUST NOT return`404 Notfound`. The *AggregatorSet*, *ToolSet*, and *CollectorSet Mix-in* collections MUST be implemented, and discovery is successful. The user MUST be allowed to associate *Mix-in*instances with the *AggregatorSet*, *ToolSet*, and *CollectorSet* collections. In an HTTP rendering, a POST over these mixins returns `200`;

5 Related works (INPUT WELCOME)

The model is reminiscent of a monitoring infrastructure that I designed and implemented in the CoreGRID EU-project [1], that in its turn is inspired by various other works (see the bibliography in the paper). The reading of the CompatibleOne prototype [2] has been enlightening concerning (among the rest) the need and possibility of modularizing the monitoring part. The 2012 revision of the OCCI core model [3] has been used as a reference.

A An example

We want to dip the Monitoring Infrastructure Management schema explained in this document into an Service Level Agreement (SLA) scenario, so let's try to define a SLA in terms of OCCI concepts.

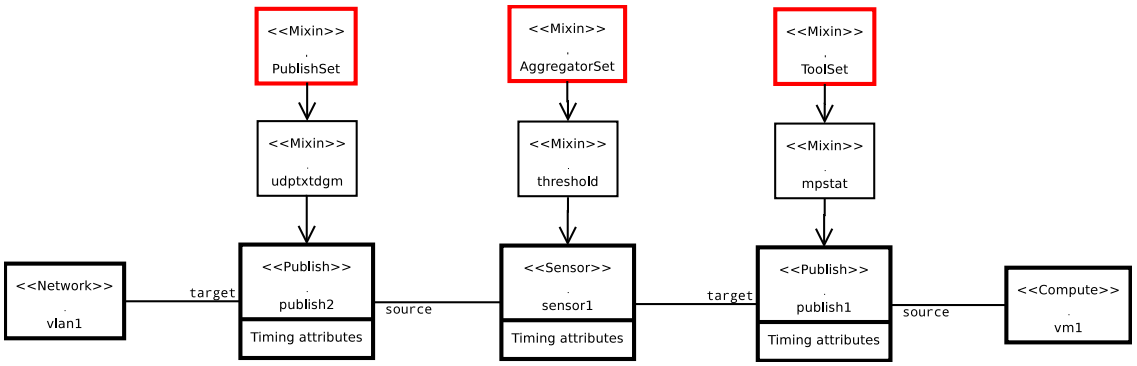

Figure 1: The instance diagram of the monitoring infrastructure

An OCCI-SLA is a contract between a user and a provider: the terms of the contract are in a form that may be provider-independent, and they are published as an OCCI-Resource in a specific namespace "occi/#sla" possibly refined with mixins. There are two basic flavors for a SLA contract:

- The provider offers a SLA: the providers offers the user the ability to monitor the conformance to SLA contract
- The user offers a SLA: the provider offers the User the tools to implement resource monitoring to meet internal SLA requirements.

Both of them are compatible with the monitoring infrastructure management schema illustrated in this paper, but are otherwise quite different.

The Service Level Agreement is an aggregate of many *Resource* that describe financial, administrative, security aspects and much more. Among such *Resource* there are the Service Objectives (SLO). Their function is to specify the meaning of "quality of service" for the specific infrastructure. This concept is translated in a function of system parameters of operation, or metrics. The SLA resource contains the instructions to associate an action to a given SLO pattern.

A user that wants to instantiate a monitoring infrastructure starts from identifying the Resources and the metrics of interest. Next the basic monitoring infrastructure is instantiated, assembling generic *Sensor Resource*s and *Collector Link*s. The following step consists of browsing the *ToolSet Mix-in* collection finding a *Mix-in*that offers the right metrics, and the first stage *Collector Link* is associated with it. Note that a given monitoring technology may require more than one *Collector Link* to operate (e.g., consider iperf). Another *Mix-in* for the *Sensor Resource* is discovered inside the *AggregatorSet* collection, and the *Sensor Resource* is associated with it. Finally, a publishing technology is selectd from the *CollectorSet Mix-in* collection, and the second stage *Collector Link* is associated with it.

Note that the first stage collector is not associated with a *CollectorSet Mix-in*, since the coupling between the *Sensor Resource* and the monitored *Resource* is managed internally, while the second stage collector is not associated with a *Mix-in* in the *ToolSet* collection since it has no monitoring activity.

The following example gives a more detailed insight of the process: it illustrates a *Sensor Resource* that measures processor utilization for a given virtual machine `vm1`, and triggers an alarm when the idle time becomes less than 10%. The alarm message is pushed as a UDP packet injected in a VLAN. We refer to the HTTP rendering to give a better insight of the operation. The object diagram is in figure 1

The user starts instantiating a new *Sensor Resource*, and a *Collector Link* connecting `vm1` to the sensor. The new *Sensor Resource*:

```
> POST /sensor/ HTTP/1.1
> Category: sensor;
            scheme:"http://schemas.ogf.org/occi/monitoring#";
            class="kind"
```

```
...
< HTTP/1.1 201 OK
< Location: "http://provider.com/monitoring/sensor1
```

the input *Collector Link*:

```
> POST /collector/ HTTP/1.1
> Category: collector;
>           scheme="http://schemas.ogf.org/occi/monitoring#";
>           class="kind";
> X-OCCI-Attribute: occi.core.target="http://provider.com/monitoring/sensor1
> X-OCCI-Attribute: occi.core.source="http://provider.com/vms/vm1
> ...
...
< HTTP/1.1 201 OK
< Location: "http://provider.com/monitoring/collector1
```

and the input *Collector Link*:

```
> POST /collector/ HTTP/1.1
> Category: collector;
>           scheme="http://schemas.ogf.org/occi/monitoring#";
>           class="kind";
> X-OCCI-Attribute: occi.core.target="http://provider.com/net/vlan1
> X-OCCI-Attribute: occi.core.source="http://provider.com/monitoring/sensor1
> ...
...
< HTTP/1.1 201 OK
< Location: "http://provider.com/monitoring/collector2
```

The timing attribuites of the three instances are filled in

```
POST /monitoring/sensor1/ HTTP/1.1
> ...
> X-OCCI-Attribute: occi.sensor.period=10;
> X-OCCI-Attribute: occi.sensor.periodspec="1,0.1,1";
> X-OCCI-Attribute: occi.sensor.timestart=10
> X-OCCI-Attribute: occi.sensor.timestop=3600;
> X-OCCI-Attribute: occi.sensor.timegranularity="1,0.1,1";
```

```
POST /monitoring/collector1/ HTTP/1.1
> ...
> X-OCCI-Attribute: occi.collector.period=10;
> X-OCCI-Attribute: occi.collector.periodspec="1,0.1,1";
```

```
POST /monitoring/collector2/ HTTP/1.1
> ...
> X-OCCI-Attribute: occi.collector.period=10;
> X-OCCI-Attribute: occi.collector.periodspec="1,0.1,1";
```

The monitoring activity will start in 10 seconds and last for 1 hour, performing one measurement every 10 seconds. Granularity and accuracy are just consistent with the timing requirements.

Next, the user browses the `ToolSet` collection looking for a tool that measures processor idle time: the search pattern comes from outside our scenario. It finally finds `mpstat`, defined as in table 4 in the provider's namespace `http://provider.com/monitoring/`.

Then it associates `link1` with the `mpstat` *Mix-in*:

```
> POST /toolset/mpstat/ HTTP/1.1
> X-OCCI-Location: http://provider.com/monitoring/collector1
```

This latter operation is critical, and may give rise to a number of errors, that result in 4xx and 5xx error codes. For instance, the server may return `403 Forbidden` in the case the `MonitoringTool` *Mix-in*is not legal for the target resource.

Model attribute	value
scheme	http://provider.com/monitoring#
term	mpstat
attributes	(see table below)

Set of Attributes for the *mpstat* Mix-in				
name	type	mut.	req.	Description
com.provider.tool.port	number	yes	yes	The port where to send a measurement trigger (control)
com.provider.tool.ncpu	number	no	yes	The number of processors (metric)
com.provider.tool.idletimecpu	number	no	yes	Total percent of idle time (metric)
com.provider.tool.usertimecpu	number	no	yes	Total percent of user time (metric)
com.provider.tool.systimecpu	number	no	yes	Total percent of system time (metric)

Table 4: Attributes defined for the mpstat mixin

Model attribute	value
scheme	http://provider.com/monitoring#
term	threshold
attributes	(see table below)

Set of Attributes for the *threshold*				
name	type	mut.	req.	Description
com.provider.sensor.threshold	number	yes	yes	The threshold value (control)
com.provider.sensor.mode	Once,Continuous	yes	yes	How frequent the warning message (control)
com.provider.sensor.fallmsg	String	yes	yes	The falling edge message
com.provider.sensor.risemsg	String	yes	yes	The rising edge message
com.provider.sensor.input	URI	no	yes	The input value (input)

Table 5: Attributes defined for the `threshold` mixin

In the general case, the above steps are repeated for every metric that the user needs to measure to compute the application-denpendent metric. Here we proceed to the next step.

The user now searches a *Mix-in* in the `AggregatorSet` collection that returns a threshold signal: it finds the `Threshold` defined in table 5

The next step of the user is to associate the *Sensor Resource* to the *Mix-in*,

```
> POST /computetool/threshold/ HTTP/1.1
> ...
> X-OCCI-Location: http://provider.com/monitoring/sensor1
```

and fills in the attributes as appropriate:

```
POST /monitoring/sensor1/ HTTP/1.1
> ...
> X-OCCI-Attribute: com.provider.sensor.threshold=10
> X-OCCI-Attribute: com.provider.sensor.mode="Once"
> X-OCCI-Attribute: com.provider.sensor.fallmgs="Warning: vm1 overloaded"
> X-OCCI-Attribute: com.provider.sensor.risemgs="vm1 load below 90%"
> X-OCCI-Attribute: com.provider.sensor.input="com.provider.monitoring.tool1.idletimecpu"
```

The server here responds with a `404 Not found` if the `input` attribute does not exist, or `401 Unauthorized` if the user is not allowed to operate on that *Resource*(e.g., the metric is outside its scope).

Finally the user associates a way to publish the result: a UDP datagram on a network. It looks in the `CollectorSet` collection the *Mix-in* that applies, and finds the one described in figure 6, that sends a string as a UDP datagram.

It then associates that *Mix-in*to the outgoing *Collector Link*:

```
> POST /collectorset/udptxtdgm/ HTTP/1.1
> ...
> X-OCCI-Location: http://provider.com/monitoring/collector2
```

and fills in the attributes as appropriate:

```
POST /monitoring/collector2/ HTTP/1.1
> ...
> X-OCCI-Attribute: com.provider.schema.destination="ctr1.provider.com";
> X-OCCI-Attribute: com.provider.schema.port="10222";
> X-OCCI-Attribute: com.provider.schema.mode="nonempty";
> X-OCCI-Attribute: com.provider.schema.input="http://provider.com/monitoring/sensor1/compoutput";
```

Model attribute	value
scheme	http://provider.com/monitoring#
term	udptxtdgm
attributes	(see table below)

Set of Attributes for the *udptxtdgm*				
name	type	mut.	req.	Description
com.provider.sensor.dest	String	yes	yes	The destination of the message (control)
com.provider.sensor.port	number	yes	yes	The destination port (control)
com.provider.sensor.mode	all,nonempty	yes	yes	Indicate whether only non empty msg are sent (control)
com.provider.sensor.input	URI	yes	yes	The msg to be sent

Table 6: Attributes defined for the `udptxtdgm` mixin

References

[1] Augusto Ciuffoletti, Yari Marchetti, Antonis Papadogiannakis, and Michalis Polychronakis. Prototype implementation of a demand driven network monitoring architecture. In *Proceedings of the CoreGRID Integration Workshop*, Hersonissos (Greece), April 2008. Available through www.slideshare.net.

[2] Iain James Marshall and Jean-Pierre Laisn. *CompatibleOne Resource Description System*, 2012.

[3] Open Grid Forum. *Open Cloud Computing Interface - Core*, June 2011. Available from www.ogf.org.

[4] Open Grid Forum. *Open Cloud Computing Interface - Infrastructure*, June 2011. Available from www.ogf.org.