GWD-C-P
OCCI-WG
augusto@di.unipi.it

Augusto Ciuffoletti, Dept. Of Computer
Science - Univ. of Pisa
February 2013

Describing a monitoring infrastructure with an OCCI-compliant schema

Status of This Document

Group Working Draft (GWD)

Document Change History

February 1st 2013: first revision (Augusto Ciuffoletti)

Copyright Notice

Trademark

Abstract

This document *provides information* to the Grid community about resource monitoring. It *describes* an OCCI Extension that allows to inspect the operation of functional resources; the provision of this API is considered as optional for the provider.

This document *presents* 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 need to be specialized using OCCI *Mix-in*s. Using this API, the user is provided with a monitoring infrastructure *on demand*.

This document does not define any standards or technical recommendations.

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

Contents

1 Introduction

This document describes an interface to define a monitoring infrastructure. It is based on the concepts introduced by OCCI, it is intended to be a first step towards the definition of a protocol to manage and verify Service Level Agreement (SLA), not being limited to SLA.

The purpose of this specification is that of giving the user the possibility to arrange a monitoring infrastructure in the way that best suits user's needs, instead of limiting the user to the implicit monitoring provided by a SLA. 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 specifically 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, and therefore to perform specific 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 activity: the specification schema must give the user the possibility 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 [OGF(2011a)] 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 [OGF(2011b)]): 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 Resource*. Appropriate *Mix-in*s associated with the *Collector Link* describe the logging activity and the database access mode.

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

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*".

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

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.

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 The *Sensor Resource*

The *Sensor Resource* (see table 2) 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 ob-

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	mutable	required	Description
occi.sensor.period	number	true	true	The time between two following measurements
occi.sensor.periodspec	string	true	false	granularity, accuracy, exponent of period measument
occi.sensor.timebase	number	false	true	The server time when the timestart and timestop are modified
occi.sensor.timestart	number	true	true	The delay after which the session is planned to start
occi.sensor.timestop	number	true	true	The delay after which the session is planned to stop
occi.sensor.timespec	string	true	false	granularity, accuracy, exponent of time measurement

Table 2: Definition of the *Sensor Resource* Kind

servations are produced, and by the scheduling times of its operation. The attributes with `required=true` 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 of rate measurement, and the floating point exponent. By default `periodspec="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`.

2.2 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:

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	mutable	required	Description
occi.collector.period	number	true	true	The time between two following measurements
occi.collector.periodspec	string	true	false	granularity, accuracy, exponent of period measument

Table 3: Definition of the *Collector Link* Kind

- 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 its specification falls outside the scope of this document.

3 The *Collector Link*

The *Collector Link* models (see table 3) the transfer of metric measurements from one *Resource* to another. The transfer 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 considered as 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. 2.1.

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="true",required="false",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 groups:

- Input attributes: their value MUST correspond to the name of one of the output parameters of the source *Sensor Resource*.
- Control attributes: they determine 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.

3.3 Constraints on instances

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

- 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*

4 Conformance profiles

The definition of conformance profiles is appropriate because the provision of an interface for the management of a monitoring infrastructure is optional.

Profile 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 *Mix-in* returns `404 Notfound`;

Profile 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 or a GET over these *Resource* collections return 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 MUST 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`;

Profile 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

The model is reminiscent of a monitoring infrastructure that I designed and implemented in the CoreGRID EU-project [Ciuffoletti et al.(2008)Ciuffoletti, Marchetti, Papadogiannakis, and Polychrona that in its turn is inspired by various other works (see the bibliography in the paper). The reading of the CompatibleOne prototype [Marshall and Laisné(2012)] has been enlightening concerning (among the rest) the need and possibility of modularizing the monitoring part. The 2012 revision of the OCCI core model [OGF(2011a)] has been used as a reference.

6 Security Considerations

The API described in this document relies on the same mechanism as the basic OCCI API, of which it is an extension. In its turn, the OCCI API is designed according with a RESTFul model, a style of exposing a web service to the users.

The way this API is exposed inherits the security aspects of the RESTFul model, that can be summarized as follows:

- the web site MUST be protected to allow access only to authorized users, and to protect the content of the communication;
- the content uploaded on the web site by the user (using POST) MUST be protected;

- the content cached on third party sites not directly accessible by the user and by the provider (proxies etc.) MUST be protected.

We stress that these security warnings are shared with any ReStFul API.

The provider must ensure that a user defined *Mix-in* does not compromise the security of other services. The provider may attain this by restricting the functionalities associated to a *Mix-in* (the limit case is the provision of templates) or run the functionalities associated to a *Mix-in* in a protected environment (e.g., as a Unix user in a chroot jail). This issue is shared with the OCCI model.

Concerning the kind of monitoring infrastructure deployed using the *Sensor Resource* and the *Collector Link*, security aspects are managed using appropriate *Mix-in*s. For instance the *Collector Link* might be associated with a *Mix-in* describing a secure transport protocol, while the sensor might be configured to be accessible only from authenticated users (?). The provider SHOULD offer the user a set of predefined *Mix-in*s that introduce the appropriate level of security. User defined *Mix-in*s SHOULD be avoided for this kind of options.

7 Glossary

metric a metric is a mathematical representation of a well defined aspect of a physical entity

measurement a measurement is the process of extracting a metric from a physical entity, and by extension also the result of such process. The measurement seldom corresponds exactly to the value of the metric.

SLA *"An agreement defines a dynamically-established and dynamically managed relationship between parties. The object of this relationship is the delivery of a service by one of the parties within the context of the agreement."* from *SLA@SOI Glossary*

Restful model *"REST is a coordinated set of architectural constraints that attempts to minimize latency and network communication, while at the same time maximizing the independence and scalability of component implementations."* [Fielding and Taylor(2002)]

OCCI "*The Open Cloud Computing Interface (OCCI) is a RESTful Protocol and API for all kinds of management tasks. OCCI was originally initiated to create a remote management API for IaaS model-based services, allowing for the development of interoperable tools for common tasks including deployment, autonomic scaling and monitoring"* [OGF(2011a)]

OCCI *Kind* *"The Kind type represents the type identification mechanism for all Entity types present in the model"* [OGF(2011a)]

OCCI Link *"An instance of the Link type defines a base association between two Resource instances."* [OGF(2011a)]

OCCI *Mix-in* *"The Mixin type represent an extension mechanism, which allows new resource capabilities to be added to resource instances both at creation-time and/or runtime."* [OGF(2011a)]

OCCI *Resource* *"A Resource is suitable to represent real world resources, e.g. virtual machines, networks, services, etc. through specialisation."* [OGF(2011a)]

Sensor Resource The *Sensor Resource* is a *Resource* that collects metrics from its input side, and delivers aggregated metrics from its output

Collector Link The *Collector Link* is a link that conveys metrics: it defines both the transport protocol and the conveyed metrics.

8 Contributors

Augusto Ciuffoletti (corresponding author)
Dept. of Computer Science
L.go B. Pontecorvo - Pisa
Italy
Email: augusto.ciuffoletti@gmail.com

Andrew Edmonds
Institute of Information Technology
Zürich University of Applied Sciences
Zürich
Switzerland
Email: andrew.edmonds@zhaw.ch

Metsch, Thijs
Intel Ireland Limited
Collinstown Industrial Park
Leixlip, County Kildare, Ireland Email: thijsx.metsch@intel.com

Ralf Nyren
Email: ralf@nyren.net

Appendix - 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.

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*

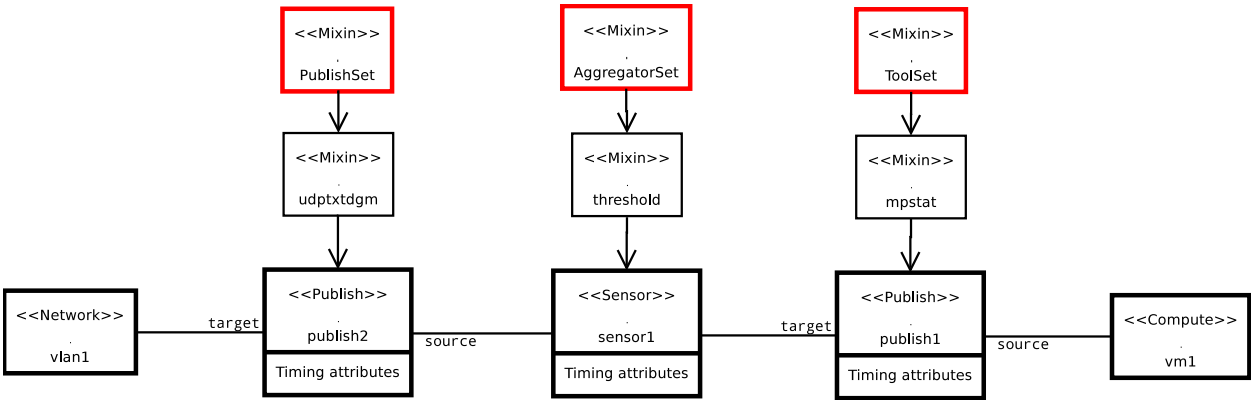

Figure 1: The instance diagram of the monitoring infrastructure

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
```

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

Set of Attributes for the *mpstat* Mix-in				
name	type	mutable	required	Description
com.provider.tool.port	number	true	true	The port where to send a measurement trigger (control)
com.provider.tool.ncpu	number	false	true	The number of processors (metric)
com.provider.tool.idletimecpu	number	false	true	Total percent of idle time (metric)
com.provider.tool.usertimecpu	number	false	true	Total percent of user time (metric)
com.provider.tool.systimecpu	number	false	true	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	mutable	required	Description
com.provider.sensor.threshold	number	true	true	The threshold value (control)
com.provider.sensor.mode	Once,Continuous	true	true	How frequent the warning message (control)
com.provider.sensor.fallmsg	String	true	true	The falling edge message
com.provider.sensor.risemsg	String	true	true	The rising edge message
com.provider.sensor.input	URI	false	true	The input value (input)

Table 5: Attributes defined for the `threshold` mixin

```
> 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 `ToolSet` *Mix-in* is not legal for the target resource.

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
> ...
```

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

Set of Attributes for the *udptxtdgm*

name	type	mutable	required	Description
com.provider.sensor.dest	String	true	true	The destination of the message (control)
com.provider.sensor.port	number	true	true	The destination port (control)
com.provider.sensor.mode	all,nonempty	true	true	Indicate whether only non empty msg are sent (control)
com.provider.sensor.input	URI	true	true	The msg to be sent

Table 6: Attributes defined for the `udptxtdgm` mixin

```
> 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/co
```

A Intellectual Property Statement

The OGF takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology

described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification can be obtained from the OGF Secretariat.

The OGF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights which may cover technology that may be required to practice this recommendation. Please address the information to the OGF Executive Director.

B Disclaimer

This document and the information contained herein is provided on an "As Is" basis and the OGF disclaims all warranties, express or implied, including but not limited to any warranty that the use of the information herein will not infringe any rights or any implied warranties of merchantability or fitness for a particular purpose.

C Full Copyright Notice

D References

[Ciuffoletti et al.(2008)Ciuffoletti, Marchetti, Papadogiannakis, and Polychronakis] 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. URL `http://www.di.unipi.it/~augusto/papers/cur_2008_a.pdf`. Available through www.slideshare.net.

[Fielding and Taylor(2002)] Roy T. Fielding and Richard N. Taylor. Principled design of the modern web architecture. *ACM Trans. Internet Technol.*, 2(2):115–150, May 2002. ISSN 1533-5399. doi: 10.1145/514183.514185. URL `http://www.cs.helsinki.fi/group/java/s12-wepa/resurssit/principled-design-of-the-modern-web-architecture.pdf`.

[Marshall and Laisné(2012)] Jamie Marshall and Jean-Pierre Laisné. *CompatibleOne Resource Description System*, 2012. URL `http://www.compatibleone.org/bin/download/Download/Software/CordsReferenceManualV2.11.pdf`.

[OGF(2011a)] OGF. *Open Cloud Computing Interface - Core*. Open Grid Forum, June 2011a. URL `http://ogf.org/documents/GFD.183.pdf`. Available from www.ogf.org.

[OGF(2011b)] OGF. *Open Cloud Computing Interface - Infrastructure*. Open Grid Forum, June 2011b. URL `http://ogf.org/documents/GFD.184.pdf`. Available from www.ogf.org.