Describing a monitoring infrastructure with an OCCI-compliant schema

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

September 20, 2012

1 Introduction

This document describes the structure of a monitoring infrastructure using the OCCI restful model. It is intended as a first step towards the definition of a protocol to manage and verify Service Level Agreement.

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 cannot be generalized. 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 describe the structure of a monitoring infrastructure, but without going into details that depend on a specific environment.

The OCCI Core Model [2] is well suited for the task, since it embeds the tools needed to extend a framework with provider specific details: this enables the description of the abstract model, leaving to a further specification the task of making explicit the details, targeting a specific provider or technology. Given that this document abstracts from the details that characterize specific infrastructure resources, the model can be applied in environments other than computing infrastructures.

The concept that makes the model extensible is the **mix-in**, in the way it is defined in OCCI: it is used to specify only the basic features of a concept, leaving the option to refine its definition with further provider or application specific details. In OCCI terms, we will define *Resources* and *Links* omitting those *Attributes* that are considered as provider or application specific. We consider that the *Mix-ins* including specific features are going to be *related* to those abstract *Resources*, and described in separate schemas. For instance, we define a `Tool` *Resource* to represent a generic monotoring tool, that is later associated with a `Ping` *Mixin* in a network monitoring specific framework.

The *mix-in* abstraction is also used to assemble custom collections of entities that form a more complex one.

The basic category defined in this document is the `OCCI-Sensor`: it represents an activity that aims at collecting a group of metrics that characterize an aspect of a computation, and at delivering them to the interested parties. For instance, an `OCCI-Sensor` may focus on the operation of a shared storage, monitoring the seek time of the disks, the frequency of read errors, the workload of the server, and the performance of the related network trunk. Since it is a collection of many simpler activities, the `OCCI-Sensor` is defined as a *Mix-in*. Its components may vary in time, according with operational conditions of the monitored infrastructure. We use two sub-types of *Resource*s to define them: the `OCCI-Tool`, and the `OCCI-Filter`. They are related with the `OCCI-Sensor` with a composition relation, and their operation is triggered according with `OCCI-Sensor` parameters.

The `OCCI-Tool` represents a specific monitoring tool, whose details are moved into a related *Mixin*, for instance a `Ping`. The *Mixin* that defines the specific tool (`Ping` in the example detailed

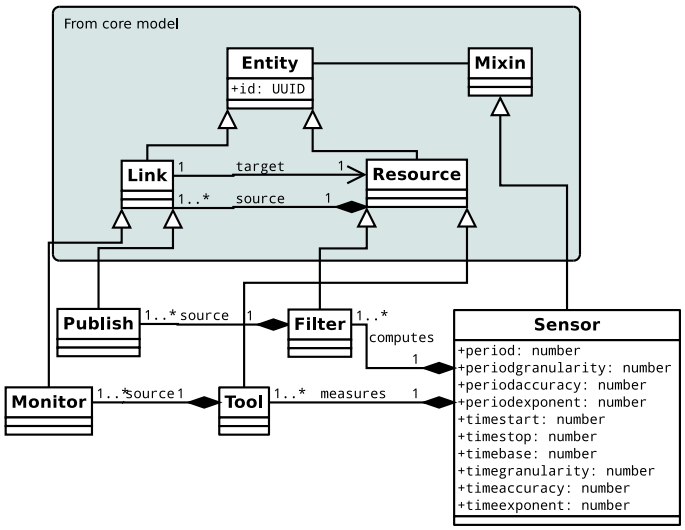

Figure 1: The UML model of the infrastructure monitoring service

Model attribute	value
scheme	http://ogf.schemas.sla/occi/monitoring#
term	sensor
attributes	(see table 2)

Table 1: Model attributes of the OCCI-Sensor mixin

in table 5) has attributes that define the input parameters that control the operation of the tool, and the resulting output measurements.

To bind the `OCCI-Tool` to the monitored *Resources* in the infrastructure we use a *Link* entity named `OCCI-Monitor`. The source of an `OCCI-Monitor` instance is an `OCCI-Tool` instance, while the target is a *Resource* in the monitored infrastructure.

The `OCCI-Filter` and the `OCCI-Publish` represent the way results produced by the `OCCI.Tools` are processed and delivered. The `OCCI-Filter` is a *Resource* that operates on the measurements to produce another set of values, and the `OCCI-Publish` is a *Link* that specifies the modalities used to publish such values. The `OCCI-Filter` embeds a script that takes as parameters the values measured by the `OCCI.Tools`, and assigns new values to the local attributes of the `OCCI-Filter`. An `OCCI-Filter` may have a memory of past measurements, while the `OCCI-Tool` is considered as stateless. The `OCCI-Publish` link takes the values of the attributes produced by the trigger, and delivers them as specified: its operation is controlled by *OCCI-Sensor* parameters as well.

Note that the schema is transparent to the existence of a standard for the naming of the measured metrics: 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.

The UML view of the overall model is described in figure 1.

2 Performance sensors for OCCI-Resources

An `OCCI-Sensor` is defined as a *Mixin*: it is a collection of `OCCI-Tools` and `OCCI-Filters` that implement its activity. In table 1 there is a summary of its model attributes.

An `OCCI-Sensor` is characterized by attributes that define the rate with which new observations are produced, and by the scheduling times of its operation.

name	type	mut.	req.	def.	Description
occi.sensor.period	number	yes	yes	-	The time between two following measurements
occi.sensor.periodgranularity	number	yes	no	-	The granularity of period measurement
occi.sensor.periodaccuracy	integer	yes	no	-	The accuracy with which the period is measured
occi.sensor.periodexponent	integer	no	yes	0	The base-10 exponent used to convert to seconds
occi.sensor.timestart	integer	yes	yes	-	The delay after which the session is planned to start, or (did) start
occi.sensor.timestop	integer	yes	yes	-	The delay after which the session is planned to stop or (did) stop
occi.sensor.timebase	integer	no	yes	-	Unix seconds
occi.sensor.timegranularity	integer	no	no	-	The granularity of period measurement
occi.sensor.timeaccuracy	integer	no	no	-	The accuracy with which the period is measured
occi.sensor.timeexponent	integer	no	yes	0	The server clock of last POST on timestart

Table 2: OCCI Attributes defined for the sensor mixin

The execution rate is defined using three attributes: the rate itself, the granularity with which the rate is measured, and the accuracy of rate measurement. Such attributes must not be confused with similar attributes that may characterize the measurement.

The execution of the sensor is controlled by two attributes, that describe the scheduling of sensor activity: to schedule the execution of the sensor the user modifies the starttime with a value indicating how far in the future the OCCI-Sensor is going to start its activity. The server keeps an attribute corresponding to the reference time of the start time.

All time values are represented as numbers. Except for the timebase, that is expressed as Unix seconds, all timing values use a floating point notation. The "exponent" attribute is the base-10 conversion that has to be used to obtain a value in seconds. A distinct exponent is used for period and scheduling purposes.

2.1 The monitoring tool

An instance of a monitoring tool is used to describe the capabilities of a given sensor: this is obtained associating one or more `OCCI-Tool`s to an `OCCI-Sensor` instance. A monitoring tool MUST contain a `related` attribute pointing to an `OCCI-Sensor` instance.

A monitoring tool is characterized by a number of specific attributes that control its operation: for instance, a ping tool is controlled by an attribute that controls the size of the exchanged packet. In addition the OCCI-Tool MUST contain one attribute for each metric it can deliver, whose last observed value is recorded in the attribute itself.

To introduce the needed flexibility the `OCCI-Tool` is defined as a *mix-in*: in our example, the length of the packet will be described in a specific `Ping` mixin related to an `OCCI-Tool`. For discovery purpose, `OCCI-Tool` mixins MUST be grouped in a `ToolSet` *Mixin* defined in the provider space (see figure 2).

In table 3 there is the list of model attributes for an `OCCI-Tool`.

In table 4 we examplify a ping mixin defined in provider's space: in our perspective, other providers may define a functionally identical plugin using different identifiers, or use the same identifiers for a different meaning. The attributes other than those that are inherited are in table 5.

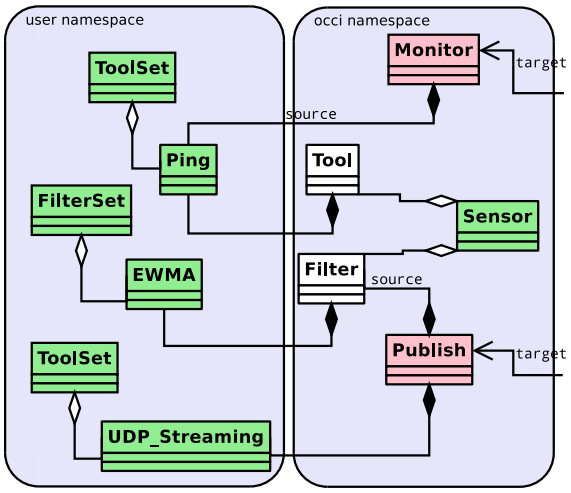

Figure 2: The class diagram with a provider that offers a "ping" monitoring tool, a Exponentially Weighted Moving Average filter, and a UDP streaming publication facility. Attributes have been omitted, and the user and occi namespaces have been highlighted. Green classes are Mixins, red classes are Links, and white classes are Resources.

Model attribute	value
scheme	http://ogf.schemas.sla/occi/monitoring#
term	tool
attributes	-
related	http://ogf.schemas.sla/occi/core#resource

Table 3: Model attributes of the OCCI-Tool resource

Model attribute	value
scheme	http://schemas.provider.com/monitoring#
term	ping
attributes	(see table 5)

Table 4: Model attributes of the Ping mixin

name	type	mut.	req.	def.	Description
com.provider.schema.ping.period	number	yes	yes	1	The time between two following measurements (in seconds)
com.provider.schema.ping.packetsize	number	yes	yes	56	The size of the probe packet (in bytes)
com.provider.schema.ping.timestamp	number	yes	yes	-	The timestamp (seconds) of the last observation
com.provider.schema.ping.roundtrip	number	no	yes	-	The last roundtrip time (in msecs)
com.provider.schema.ping.packetloss	number	no	yes	-	The frequency of packet loss events (percent)

Table 5: Attributes defined for the Ping mixin

Model attribute	value
scheme	http://ogf.schemas.sla/occi/monitoring#
term	monitor
source	URI
target	URI
attributes	-
related	http://ogf.schemas.sla/occi/core#link

Table 6: Attributes of the OCCI-Monitors link

Model attribute	value
scheme	http://ogf.schemas.sla/occi/monitoring#
term	filter
attributes	-
related	http://ogf.schemas.sla/occi/core#resource

Table 7: Model attributes of the OCCI-Filter resource

A `OCCI-Tool` instance usually operates on one or more *Resource*s of the monitored infrastructure, for instance a `Compute` *Resource*. This relationship is represented with *Links* that are subtypes of the `OCCI-Monitors` *Link* entity. `OCCI-Tool`s are characterized by a number of related resources that depends on their functionality. For instance, an `OCCI-Tool` representing a ping session may originate two links, one for each of the network interfaces involved in the measurement, whereas a tool measuring the effectiveness of a load balancing service may be linked to a variable number of nodes.

An `OCCI-Monitor` instance has its source on a `OCCI-Tool`, and its target on a monitored resource: it is defined as in table 6. The model attributes consist of a source and a target, and the provider may define mixins to enrich the basic type.

The soundness of the `OCCI-Link`s originating from an `OCCI-Tool` — integrated with a *Mixin* — with respect to the resulting activity can be verified only at runtime: for instance, a ping tool has to be linked to two resources, but in the case that only one link exists, the problem is detected, and an error is generated, only when the tool starts its operation. This is a limit of the current scheme: it would be preferable to have a way to make a static check, but this is apparently inconsistent with the need of a dynamic configurability.

2.2 Measurement aggregation and inference

As a general rule, data made available by the Tools have to be processed before being delivered to the user: this happens, for instance, when a flag has to be raised whenever a number of conditions hold, or when the user wants a robust indicator (like an EWMA) instead of a instantaneous measuremnt. For this reason an `OCCI-Sensor` MAY be associated with a functionality that processes the measurements collected by the `OCCI-Tool`s: this component is called *OCCI-Filter*, and an `OCCI-Sensor` may contain several distinct `OCCI-Filters`. Each piece of data produced by an `OCCI-Filter` is made available through an attribute. To provide the needed flexibility to the functionality of the `OCCI-Filter`, it is left totally abstract, and the user creates and applies a custom *Mixin*.

The formula used to transfom raw measurements is represented by a script: the OCCI-Attributes that are used as input and output for the script are represented, inside the script, as variables. The syntax used for such association is not dealt with in this document, and depends on the scripting language. The script language should be extended so that OCCI-Sensor timing triggers appear as asynchronous manageable events.

Similarly to the tool *Mixin*s, also `OCCI-Filter`s must be grouped inside a unique `FilterSet` *Mixin* inside provider and user namespaces.

Model attribute	value
scheme	http://ogf.schemas.sla/occi/monitoring#
term	publish
attributes	-
related	http://ogf.schemas.sla/occi/core#resource

Table 8: Model attributes of the Publish link

2.3 Delivering monitoring results

The way measurements are delivered is described with a `OCCI-Publish` *Link*: for the sake of flexibility it is defined as an empty framework, that is finalized by way of *Mixin*s. For instance, one mixin may indicate the repository where measurements are made available, another may indicate that measurements are piped to a destination, another that the last measurement is available with a GET. The available mixins, as well as the associated semantics, depend on the specific provider. An `OCCI-Sensor` MUST be related with at least one `OCCI-Publish` *Link*. The list of its model attributes is in table 8.

Also in this case, a `PublishSet` *Mixin* is defined in provider's space to make discoverable the way of publishing the monitoring results.

3 Monitoring infrastructure construction and discovery

Here the OCCI prefix is omitted.

Summarizing, a user that wants to instantiate a monitoring infrastructure starts from identifying the Resources and the metrics of interest. Next it searches in the "provider/#ToolSet" the Tools that offer such metrics. Next it instantiates the Tools needed to perform the required measurements: there may be the need of several tools of the same kind (more than one "ping"). Next the Monitor links are built for each Tool instance towards the resources it monitors. A new Sensor is created, and all the tools needed to compute the measurements are linked to the new Sensor. More than one OCCI-Sensor can be instantiated if the monitoring activity is partitioned for some reason. A number of Filters is added to each Sensor, to compute the derivate metrics that are considered of interest. Finally, the Channel resource is instantiated from the namespace "provider/#PublishSet", and its attributes are configured through mixins. Whenever the sensor is activated (according with its starttime attribute) a flow of data starts to be available to the user through the defined channel.

4 Service Level Agreement

A Service Level Agreement (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.

The Service Level Agreement is an aggregate of many Resources, that describe financial, administrative, security aspects and much more. Among such Resources, 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 *standard* parameters of operation.

In order to monitor the compliance to the SLA, the SLA management infers what are the available metrics that need to be measured to compute the standard parameters of operation, and instantiates the corresponding Tools. The appropriate Filters are also instantiated to derive standard metrics from the locally available ones measured by the Tools, that represent the level of conformance of the service to the SLO. Finally, the Filters are connected to the SLA monitoring functionality through a specific Publish link. The SLA resource contains the instructions to associate an action to a given SLO pattern.

5 Related works works

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 [**?**] has been enlightening concerning (among the rest) the need and possibility of modularizing the monitoring part. The 2012 revision of the OCCI core model [2] has been used as a reference.

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.

6 References

OCCI Core

Articolo mio

compatibleone