FEDERATED TEST-BEDS FOR LARGE-SCALE INFRASTRUCTURE EXPERIMENTS
FELIX EU-JP

Collaborative joint research project co-funded by the European Commission (EU)and National Institute of Information and Communications Technology (NICT) (Japan)

Grant agreement no: 608638
Project acronym: FELIX
Project full title: "Federated Test-beds for Large-scale Infrastructure eXperiments"
Project start date: 01/04/13
Project duration: 36 months

Deliverable D3.2
Slice Monitoring

Version 1.0

Due date: 31/11/2014

Submission date: 30/01/2015

Deliverable leader: iMinds

Author list: Takatoshi Ikeda (KDDI), Carolina Fernandez (i2CAT), Carlos Bermudo (i2CAT), Bart Puype (iMinds)

Dissemination level

☑	PU:	Public
☐	PP:	Restricted to other programme participants (including the Commission Services)
☐	RE:	Restricted to a group specified by the consortium (including the Commission Services)
☐	CO:	Confidential, only for members of the consortium (including the Commission Services)

<THIS PAGE IS INTENTIONALLY LEFT BLANK>

Table of Contents

List of Figures

List of Tables

Abstract

This document details the monitoring system for the slice which is composed of heterogeneous resources in the FELIX Federated Framework. An overview of the design and implementation for the Monitoring System modules is given; as well as explaining their key internal and inter-communication workflows.

Excecutive Summary

Deliverable D3.2 explains the Monitoring System (MS), which is aimed at retrieving metrics from the slices in the FELIX Federated Framework. This document comprises the design and implementation details of the MS module, its internal components and other Monitoring Tools, as well as a brief explanation on how do some specific third-party Monitoring Tools work. It also contains information on the deployment to be performed by the administrator, as well as the common set of operations performed by the MS. This document is part of a set of deliverables that describe the implementation and design, deployment and operation. In this aspect, D3.1 can be read for better understanding how the Computing Resource Manager (C-RM) and Software-Defined Networking Resource Manager (SDN-RM) work and how the Resource Orchestrator (RO) retrieves part of the monitoring metrics; while D3.3 and D3.4 can be consulted for inter-domain networking and the end-user interfaces and tools, respectively.

We introduce first the concepts and definitions common to the different modules and tools taking part in the monitoring process. After that we explain the design and implementation of the MS and Monitoring Tools and finally, guidelines are provided for the deployment, configuration and operation of the MS module and some of the Monitoring Tools used by some components of the MS.

This document is addressed to software architects, software and network engineers, software developers implementing specific features of the MS and also system administrators.

1 Introduction

In the FELIX Federated Framework, a slice is created from a set of heterogeneous resources, each properly provisioned through their RMs and scattered among a number of islands. In such an environment, it is not easy to identify the source of a problem (for instance a resource on a given island) in order to promptly solve it; and that is so because each resource is individually monitored within its island's Monitoring System. Furthermore, the monitoring of the physical equipment is not enough; as there are resources of the slice that are being dynamically generated by the experimenter and whose status must be tracked as well.

The Monitoring System (MS) in the FELIX Framework supports the monitoring of such dynamically provisioned resources by cooperating with the Resource Orchestrator (RO), that is the module that intercepts the experimenter requests and steers them to the appropriate Resource Manager in order to plan, allocate, provision and manage the resources.

In the FELIX Framework, the MS is deployed per island, where it performs a number of tasks. Specifically, it periodically retrieves monitoring data of the FELIX infrastructure (through the Monitoring tools, further explained in following sections) and it shall also receive slice and physical topology information periodically from the RO. The MS adopts a hierarchical structure in a manner that 1) the slice and physical topologies data is aggregated per island and sent from each RO to its Master RO (MRO), which is located in the upper layer; and 2) the monitoring data is aggregated per island and sent from MS to Master MS (MMS). Once the information is aggregated into the modules of the upper layer, the FELIX Framework can access all the monitoring data of the slice, as well as the information of the physical topology. This information covers different islands and can now be interpreted and used for providing information to the experimenter and administrator of each testbed through the available Monitoring GUI.

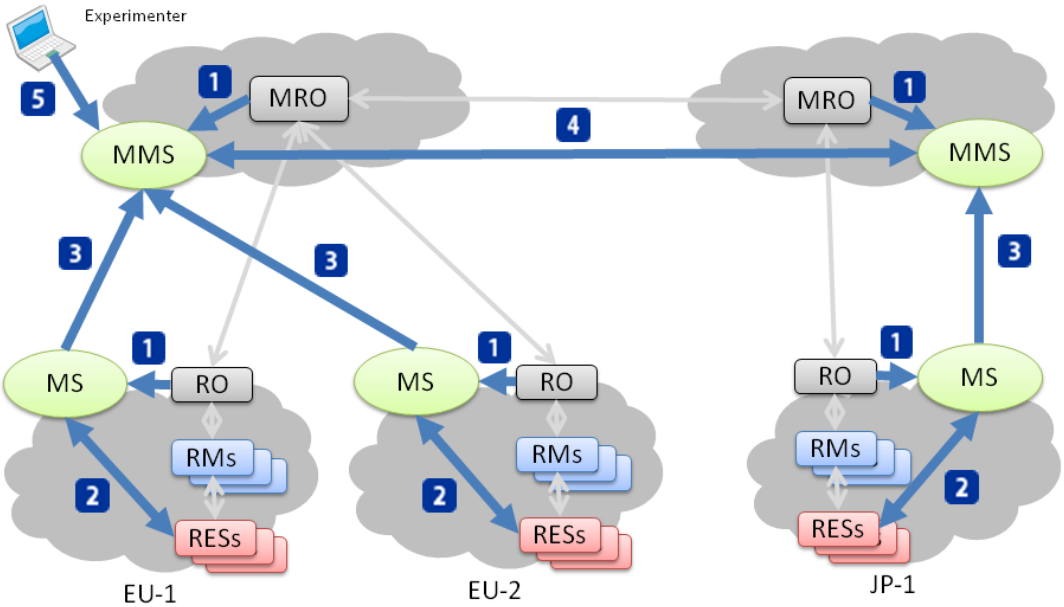

Figure 1.1: Flow of information in the Monitoring System

Figure 1.1 features a typical flow of information between the modules related to the MS. Each flow corresponds to a specific step:

1. (M)MS obtains both slice and physical topology information

- Information of the slice resources and the slice and physical topology is needed by (M)MS.
- MS retrieves information from RO (*lower layers*):
  (a) RO to retrieve this information from its RMs and examining users' requests.
  (b) RO pushes such information to MS.
- MMS retrieves information from MRO (*upper layers*):
  (a) MRO to retrieve aggregated information from its underlying ROs and examining users' requests.
  (b) The information is pushed from each RO to MRO, as specified by the grey lines in Figure 1.1.
  (c) MRO pushes such information to MMS.

2. MS initialises the monitoring process
   - Monitoring data starts to be retrieved, based on events and information pushed by RO.
3. MS forwards or aggregates monitoring data to MMS
   - Monitoring data consists of metrics per resource, typically measured from 3rd party applications.
   - MS per island forwards or aggregates the monitoring data to MMS.
   - MMS stores all monitoring data coming from subordinate MS.
4. Monitoring data exchanged between MMS
   - MMS exchanges its monitoring data with other MMS.
   - As a result, MMS are synchronised and keep all monitoring data for slices and infrastructures.
5. MS provides monitoring data to experimenters
   - MS/MMS provides its monitoring data through a specific API and the Monitoring GUI.
   - Scope of the data:
     – MS provides partial monitoring data (i.e. information of local island).
     – MMS provides whole monitoring data of FELIX islands (i.e. aggregated information of multiple islands).

In the following sections we will describe in more detail the internal design and implementation of the MS and its components; as well as the interactions between them. We also provide information of other Monitoring Tools used by the MS and finally, we document a brief guide on how to deploy and operate with the MS and related tools.

2 Definitions

Throughout this document we use specific notation and acronyms that are explained here. Please refer to this guide to identify the concept or for a more detailed explanation.

2.1 Abbreviations

- **AAA**: Authorisation, Authentication and Accounting.
- **API**: Application Programming Interface.
- **DTD**: Document Type Definition
- **GUI**: Graphical User Interface.
- **MIB**: Management Information Bases.
- **MMS**: Master Monitoring System.
- **MRO**: Master Resource Orchestrator.
- **MS**: Monitoring System.
- **OGF**: Open Grid Forum.
- **OID**: Object Identifier
- **REST**: Representational State Transfer.
- **RM**: Resource Manager.
- **RO**: Resource Orchestrator.
- **RSpec**: Resource Specification.
- **SNMP**: Simple Network Management Protocol.

2.2 Definitions

- **API**: Interface implemented by an application for the inter-communication to the other applications.
- **Hypervisor**: Piece of computer software that creates and runs virtual machines.
- **Island**: Physical domain under particular management. It provides infrastructure and resources to the end user.
- **RM**: Software component able to reserve, create, manage and delete resources by communicating with the hardware. It provides interfaces for both administrative and common operations on resources.
- **RSpec**: XML document following agreed schemas to represent resources that are understood by Resource and Aggregate Managers.

3 Implementation Details

We proceed in this section to the documentation of the **Monitoring System** (MS) module, whose main objective is to aggregate and manage both monitoring and topology (slice and physical) information, which can be used for decision-making at any point of the FELIX Framework. This module also aims to provide a detailed representation of this data through its corresponding section in the Expedient GUI.

3.1 Monitoring System

The MS module contains a number of components that work together to give an aggregated view on 1) the monitoring data and 2) topology information of the FELIX testbed facilities and experimenter slices. These two operations are performed directly by the MS and in conjunction with the RO and MRO, respectively. We focus in this section on the internal components of the MS, in charge of directly retrieving monitoring information and also interpreting topology information to combine in the form aggregated data, so it is able to provide an overview of the FELIX Framework status.

3.1.1 Design

The MS module has been introduced as a new module within the FELIX Framework. The major functionalities of this software module are as follows:

- Aggregate monitoring data for the physical infrastructure and the resources per slice.
- Request further information to physical infrastructure, if needed; providing that the MS can access the infrastructure.
- Manage the lifetime of the monitoring information.
- Represent the monitoring data in a clear format to the end users.

The following sections describe the design of the MS through the components that conform it, as well as its internal workflow and interaction with related modules belonging to the FELIX Framework. The exposed APIs are documented as well, and finally there's a brief summary of the future work planned for this module.

3.1.1.1 Building blocks

The MS module is composed of 3 components, namely the **Topology API**, **Monitoring API** and the **Monitoring Data Collector**. Figure 3.1 depicts their components and how these relate to the others.

Topology API

The component implementing the Topology API manages the resource and topology information. It exposes two different APIs, depending on the action performed (*GET*/retrieve or *POST*/send information):

- **Receiver API**: this API waits for information received from the RO. The Resource Orchestrator must retrieve information subject to be monitored, such as the physical infrastructure, the resources requested by each experimenter for a given slice and the virtual topology conformed by them. This is sent to the MS under certain circumstances (e.g. periodic or certain on-demand transmissions). This interface parses the received information and stores it on its own Topology database.
- **Sender API**: this interface provides the topology information to an external entity such as a client or a monitoring GUI. In order to make this data available, the topology database is queried for the latest physical and slice topologies, then this data is aggregated and formatted following a specific structure that the client must follow when reading this data.

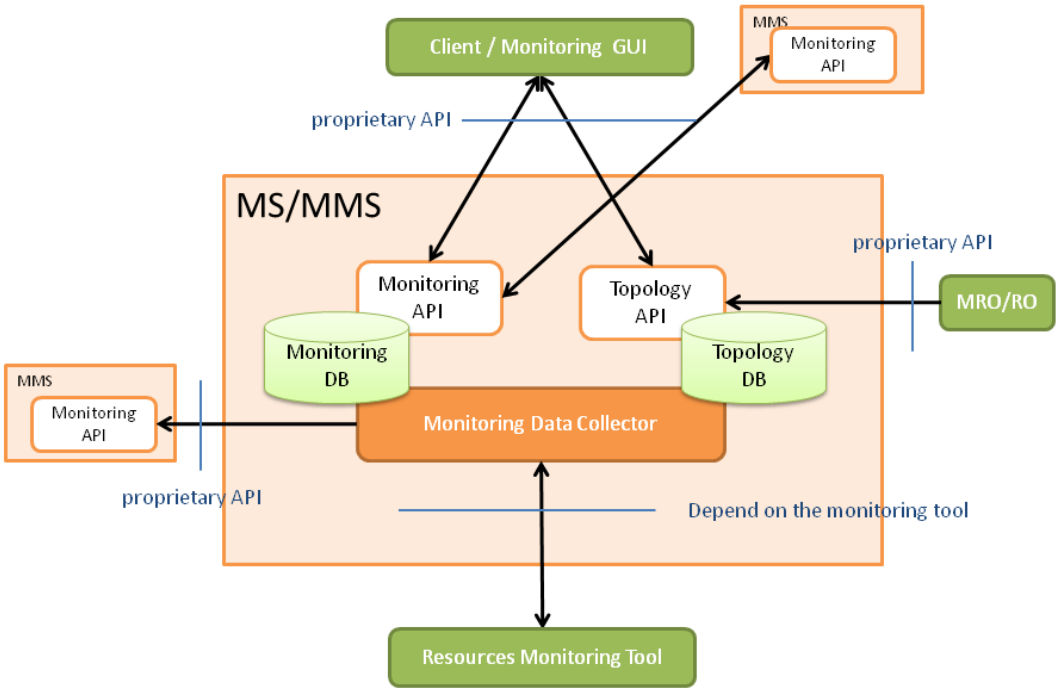

Figure 3.1: Components of the MS

Monitoring Data Collector

This component is in charge of gathering monitoring data from the physical resources in an indirect manner. This is done by interacting with third-party monitoring tools that retrieve measurements directly from the physical hardware. For instance, perfSONAR [1] and Zabbix [2] are third-party monitoring applications that contact directly the networking and computing resources. This way, the *Monitoring Data Collector* gathers the different monitoring data through the API of monitoring applications and registers this data in the Monitoring database.

Another feature of this component is to forward data (such as registration information) from the MS in the lower layer to the MMS above or to exchange information between different MMS modules. The forwarding is performed through its Monitoring API.

Monitoring API

This component is in charge of managing the monitoring data. It implements two different interfaces as well, also depending on the action performed (retrieve or send information):

- **Receiver API**: accepts incoming monitoring data and stores it into the Monitoring internal database. The *Monitoring Data Collector* component registers the monitoring data of every resource in the island into the MS. The monitoring data registered through this interface can be one of the following:
  - Physical monitoring data of every available resource.
  - Slice monitoring data, that is, data of the resources assigned to the slice.
- **Sender API**: this interface provides the monitoring data to an external entity; for instance a client or monitoring GUI.

3.1.1.2 Monitoring Data Definition

The MS tracks a set of values for each FELIX resource. That is, a resource of a given kind is associated with a

number of monitoring metrics that are to be retrieved, then registered on the internal databases and finally interpreted and used by this module.

The following tables describe the proposed metrics, grouped by their corresponding Resource Manager. Though it is not a comprehensive or final list of metrics and it may vary after the integration stage, these conform a preliminary compilation of data to be monitored per resource.

Computing Resource Manager

Metric	Description	Type	Values	Unit
Life status	Life/Death monitoring for the VM	Integer	1 = Up, 2 = Down	-
Load Average	Load Average of VM	Decimal	--	-
Memory Usage	Percentage of memory usage of VM	Integer	0-100	%
IF traffic (tx)	Traffic on network interface of VM	Integer	--	bps
IF traffic (rx)	Traffic on network interface of VM	Integer	--	bps
IF error	Error on network interface of VM	Integer	--	eps

Table 3.1: Initial monitoring metrics for the C-RM resources

Software-Defined Networking Resource Manager

Metric	Description	Type	Values	Unit
Life Status	Life/Death (and transitional statuses) monitoring for the switch	Integer	1 = Up, 2 = Down	-
Port status	Port is up/down	Integer	1 = Up, 2 = Down	-
Traffic (tx)	Transmitted-traffic rate on the network port	Integer	--	bps
Traffic (rx)	Received-traffic rate on the network port	Integer	--	bps

Table 3.2: Initial monitoring metrics for the SDN-RM resources

Stitching Entity Resource Manager

Metric	Description	Type	Values	Unit
Life Status	Life/Death (and transitional statuses) monitoring for the switch	Integer	1 = Up, 2 = Down	-
Port status	Port is up/down	Integer	1 = Up, 2 = Down	-
Traffic (tx)	Transmitted-traffic rate on the network port	Integer	--	bps
Traffic (rx)	Received-traffic rate on the network port	Integer	--	bps

Table 3.3: Initial monitoring metrics for the SE-RM resources

Transit Network Resource Manager

Metric	Description	Type	Values	Unit
Connection status	Connection (network path) is up/down	Integer	1 = Up, 2 = Down	-

Table 3.4: Initial monitoring metrics for the TN-RM resources

3.1.1.3 Exposed interfaces

As documented in the *Building blocks* section, the MS exposes 2 APIs to other modules. The structure of the URIs is previously agreed and know between the participant modules in the communication. The detailed structure of the URIs is shown in Figure 3.2.

API for client (GUI / Experimenter)

URI	description	method	from	to
http://192.168.1.1/				
monitoring-system/				
topology/	returns topology list	GET	Experimenter (GUI)	MMS
	accept topology registry	POST	MRO	MMS
			RO	MS
physical/	return physical topology info.	GET	Experimenter (GUI)	MMS
slice/ <sliceID>	return slice topology info. specified by slice ID	GET	Experimenter (GUI)	MMS
monitoring/ <RM>/	accept monitoring data of specific resource registry	POST	MMS	MMS
			MS	MMS
physical/	return physical network resource (SDN) info. specified by request message	GET	Experimenter (GUI)	MMS
slice/	return slice network resource (SDN) info. specified by request message	GET	Experimenter (GUI)	MMS

Figure 3.2: URI structure of API

On the other hand, the data that traverses this API must comply to a given structure. We provide a number

of examples below, depending on the API and the data retrieved or sent.

Topology API

MS receives from the RO monitoring information regarding both physical and slice topologies. Figure 3.3 depicts the two different kind of topologies registered in the MS, side by side, for a sample slice spanning two islands.

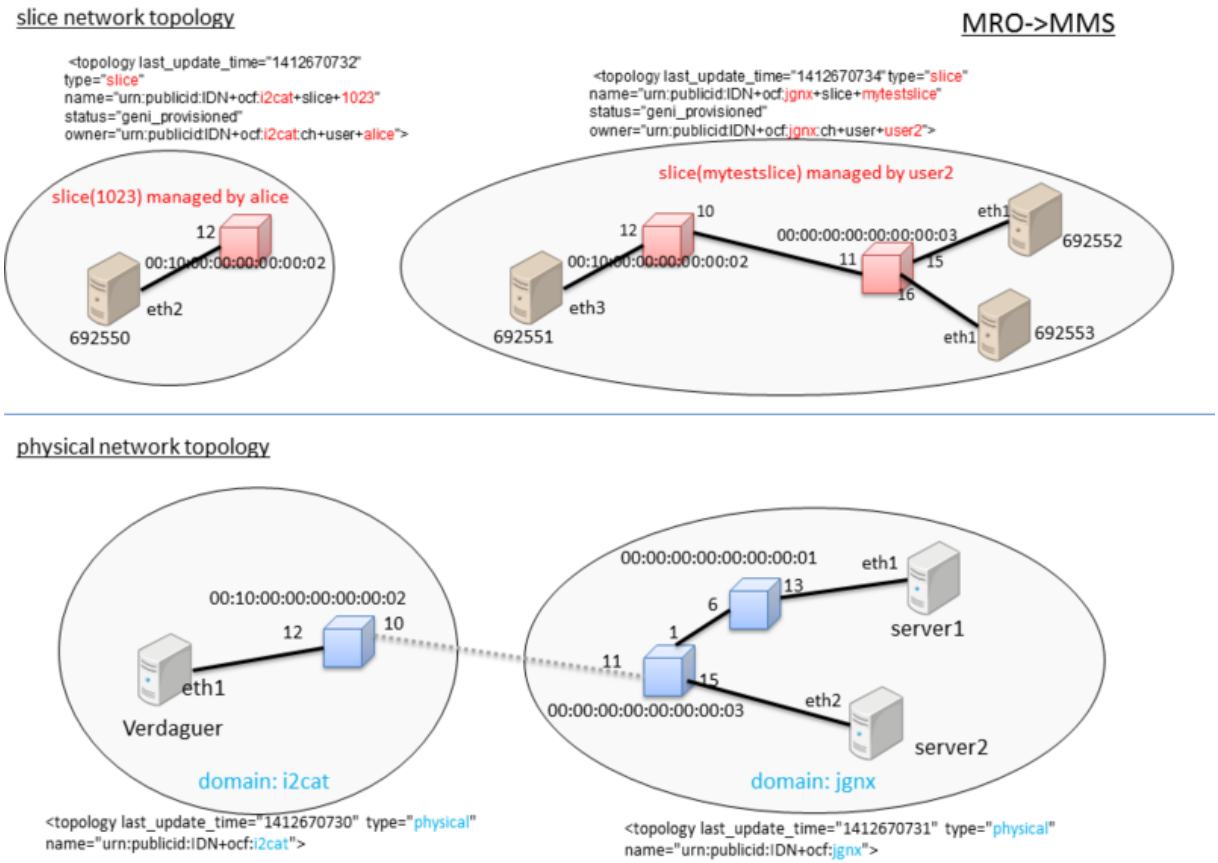

Figure 3.3: Example physical and slice topology

The figure above contains an excerpt of the topology information sent from the Resource Orchestrator module to the MS. The topology and slice information used by the MS follows each their custom XML schema, of which we provide samples below.

It is worth noting that the structure of the topology information is the same disregarding the layer at which the communication takes place (that is, either RO to MS or MRO to MMS). It is also important to state that the RO module can only retrieve topology information directly related to its domain, so the topology data in the upper-layer RO is an aggregation of that in the lower layers.

Physical topology information from MRO to MMS

The physical topology information corresponds to the lower layer in Figure 3.3. This information is transmitted from RO to the MS and also as aggregated information by their counterparts in the above layer, that is, the aggregated topology information is sent from MRO to MMS.

```
<topology_list>
 <topology last_update_time="1412670730" type="physical" name="urn:publicid:IDN+ocf:i2cat">
   <node id="urn:publicid:IDN+ocf:i2cat:vtam:Verdaguer" type="server">
     <management type="snmp">
       <address>192.168.2.1</address>
       <port>161</port>
```

```
        <auth_id>public</auth_id>
      </management>
      <interface id="urn:publicid:IDN+ocf:i2cat:vtam:Verdaguer+interface+eth1"/>
    </node>
    <node id="urn:publicid:IDN+openflow:ocf:i2cat:ofam+datapath+00:10::02"
          type="switch">
      <management type="snmp">
        <address>192.168.1.2</address>
        <port>161</port>
        <auth_id>public</auth_id>
      </management>
      <interface
          id="urn:publicid:IDN+openflow:ocf:i2cat:ofam+datapath+00:10::02_12">
          <port num="12"/>
      </interface>
      <interface
          id="urn:publicid:IDN+openflow:ocf:i2cat:ofam+datapath+00:10::02_10">
          <port num="10"/>
      </interface>
    </node>
    <link type="lan">
      <interface_ref client_id="urn:publicid:IDN+ocf:i2cat:vtam:Verdaguer+interface+eth1"/>
      <interface_ref
          client_id="urn:publicid:IDN+openflow:ocf:i2cat:ofam+datapath+00:10::02_12"/>
    </link>
  </topology>
  <topology last_update_time="1412670731" type="physical" name="urn:publicid:IDN+ocf:jgnx">
    <node id="urn:publicid:IDN+openflow:ocf:jgnx:ofam+datapath+:01" type="switch">
      <interface id="urn:publicid:IDN+openflow:ocf:jgnx:ofam+datapath+00::01_6">
          <port num="6"/>
      </interface>
      <interface id="urn:publicid:IDN+openflow:ocf:jgnx:ofam+datapath+00:00::01_13">
          <port num="13"/>
      </interface>
    </node>
    <node id="urn:publicid:IDN+ocf:jgnx:vtam:server1" type="server">
      <interface id="urn:publicid:IDN+ocf:jgnx:vtam:server1+eth1"/>
    </node>
    <link type="lan">
      <interface_ref client_id="urn:publicid:IDN+ocf:jgnx:vtam:server1+eth1"/>
      <interface_ref
          client_id="urn:publicid:IDN+openflow:ocf:jgnx:ofam+datapath+00::01_13"/>
    </link>
    <node
          id="urn:publicid:IDN+openflow:ocf:jgnx:ofam+datapath+00::03" type="switch">
      <interface id="urn:publicid:IDN+openflow:ocf:jgnx:ofam+datapath+00::03_11">
          <port num="11"/>
      </interface>
      <interface id="urn:publicid:IDN+openflow:ocf:jgnx:ofam+datapath+00::03_1">
          <port num="1"/>
      </interface>
      <interface id="urn:publicid:IDN+openflow:ocf:jgnx:ofam+datapath+00::03_15">
          <port num="15"/>
      </interface>
    </node>
    <link type="lan">
      <interface_ref
          client_id="urn:publicid:IDN+openflow:ocf:jgnx:ofam+datapath+00::01_6"/>
      <interface_ref
          client_id="urn:publicid:IDN+openflow:ocf:jgnx:ofam+datapath+00::03_11"/>
    </link>
    <link type="lan">
      <interface_ref
          client_id="urn:publicid:IDN+openflow:ocf:jgnx:ofam+datapath+00::03_15"/>
      <interface_ref
          client_id="urn:publicid:IDN+ocf:jgnx:vtam:server2+eth2"/>
    </link>
    <node id="urn:publicid:IDN+ocf:jgnx:vtam:server2" type="server">
      <interface id="urn:publicid:IDN+ocf:jgnx:vtam:server2+eth2"/>
    </node>
  </topology>
</topology_list>
```

Slice topology information from (M)RO to (M)MS

The slice topology information, in turn, corresponds to the upper layer depicted in Figure 3.3. As in the physical topology case, this information is transmitted from RO to the MS and must also be sent as aggregated information by their counterparts in the above layer, that is, the aggregated topology information must be sent from MRO to MMS.

During this transmission, all relevant data of the *mytestslice* slice information is to be recovered by the RO module and sent to MS.

```
<topology_list>
 <topology last_update_time="1412670734" type="slice"
           name="urn:publicid:IDN+ocf:jgnx+slice+mytestslice" status="geni_provisioned"
           owner="urn:publicid:IDN+ocf:jgnx:ch+user+user2">
   <node id="urn:publicid:IDN+ocf:i2cat:vtam:Verdaguer" type="server">
     <node id="urn:publicid:IDN+ocf:i2cat:vtam:Verdaguer+sliver+692551"
           type="vm">
       <management type="vm">
         <auth_user/>
         <auth_password/>
       </management>
       <interface
          id="urn:publicid:IDN+ocf:i2cat:vtam:Verdaguer+sliver+692551+interface+eth3"/>
     </node>
   </node>
   <node id="urn:publicid:IDN+openflow:ocf:i2cat:ofam+datapath+00:10::02"  type="switch">
     <interface id="urn:publicid:IDN+openflow:ocf:i2cat:ofam+datapath+00:10::02_12">
         <port num="12"/>
             <vlan start="6" end="6"/>
     </interface>
     <interface id="urn:publicid:IDN+openflow:ocf:i2cat:ofam+datapath+00:10::02_10">
         <port num="10"/>
             <vlan start="6" end="6"/>
     </interface>
   </node>
   <link type="lan">
     <interface_ref
       client_id="urn:publicid:IDN+ocf:i2cat:vtam:Verdaguer+sliver+692551+interface+eth3"/>
     <interface_ref
       client_id="urn:publicid:IDN+openflow:ocf:i2cat:ofam+datapath+00:10::02_12"/>
   </link>
   <node id="urn:publicid:IDN+openflow:ocf:jgnx:ofam+datapath+00::03"  type="switch">
     <interface id="urn:publicid:IDN+openflow:ocf:jgnx:ofam+datapath+00::03_11">
         <port num="11"/>
     </interface>
     <interface id="urn:publicid:IDN+openflow:ocf:jgnx:ofam+datapath+00::03_15">
         <port num="15"/>
     </interface>
     <interface id="urn:publicid:IDN+openflow:ocf:jgnx:ofam+datapath+00::03_16">
         <port num="16"/>
     </interface>
     <match>
       <vlan start="6" end="6"/>
     </match>
   </node>
   <link type="lan">
     <interface_ref
        client_id="urn:publicid:IDN+openflow:ocf:i2cat:ofam+datapath+00:10::02_10"/>
     <interface_ref
        client_id="urn:publicid:IDN+openflow:ocf:jgnx:ofam+datapath+00:00::03_11"/>
   </link>
   <link type="lan">
     <interface_ref
       client_id="urn:publicid:IDN+openflow:ocf:jgnx:ofam+datapath+00::03_15"/>
     <interface_ref
       client_id="urn:publicid:IDN+ocf:jgnx:vtam:server2+sliver+692552+interface+eth1"/>
   </link>
   <link type="lan">
     <interface_ref
       client_id="urn:publicid:IDN+openflow:ocf:jgnx:ofam+datapath+00::03_16"/>
     <interface_ref
       client_id="urn:publicid:IDN+ocf:jgnx:vtam:server2+sliver+692553+interface+eth1"/>
   </link>
```

```
    <node id="urn:publicid:IDN+ocf:jgnx:vtam:server2" type="server">
      <node id="urn:publicid:IDN+ocf:jgnx:vtam:server2+sliver+692552" type="vm">
        <management type="vm">
          <auth_user/>
          <auth_password/>
        </management>
        <interface id="urn:publicid:IDN+ocf:jgnx:vtam:server2+sliver+692552+interface+eth1"/>
      </node>
    </node>
    <node id="urn:publicid:IDN+ocf:jgnx:vtam:server2" type="server">
      <node id="urn:publicid:IDN+ocf:jgnx:vtam:server2+sliver+692553" type="vm">
        <management type="vm">
          <auth_user/>
          <auth_password/>
        </management>
        <interface id="urn:publicid:IDN+ocf:jgnx:vtam:server2+sliver+692553+interface+eth1"/>
      </node>
    </node>
  </topology>
</topology_list>
```

Monitoring API

The MS module accepts the monitoring data of both physical and slice resources from the *Monitoring Data Collector* component as well as from other MS modules.

Monitoring data forwarded from MS to MMS

An example of the monitoring data of the SDN resources is provided below:

```
<resource forward="yes">
  <topology type="physical">
    <node id="n1" type="switch" network_name="domainA">
      <port num="10">
        <parameter type="status">
          <value timestamp="1409204400 ?>1<value/>
          <value timestamp="1409204410">1<value/>
          <value timestamp="1409204420">2<value/>
        <parameter />
        <parameter type="in-bps">
          <value timestamp="1409204400">100<value/>
          <value timestamp="1409204410">110<value/>
          <value timestamp="1409204420">100<value/>
        <parameter />
        <parameter type="out-bps">
          <value timestamp="1409204400">200<value/>
          <value timestamp="1409204410">210<value/>
          <value timestamp="1409204420">200<value/>
        <parameter />
      </port>
      <port num="11">
        <parameter type="status">
          <value timestamp="1409204400">2<value/>
          <value timestamp="1409204410">2<value/>
          <value timestamp="1409204420">1<value/>
        <parameter />
        <parameter type="in-bps">
          <value timestamp="1409204400">100<value/>
          <value timestamp="1409204410">110<value/>
          <value timestamp="1409204420">100<value/>
        <parameter />
      </port>
    </node>
    <node id="n2" type="switch" network_name="domainA">
      <port num="10">
        <parameter type="status">
          <value timestamp="1409204300">1<value/>
          <value timestamp="1409204400">1<value/>
          <value timestamp="1409204500">1<value/>
        <parameter />
        <parameter type="in-bps">
          <value timestamp="1409204400">100<value/>
          <value timestamp="1409204410">110<value/>
```

```
          <value timestamp="1409204420">100<value/>
        <parameter />
        <parameter type="out-bps">
          <value timestamp="1409204400">200<value/>
          <value timestamp="1409204410">210<value/>
          <value timestamp="1409204420">200<value/>
        <parameter />
      </port>
    </node>
  </topology>
</resource>
```

Monitoring data exchanged between MMS modules

Similar to the example above, the following consists of the aggregated monitoring data that was previously collected from different MS modules.

```
<monitoring-data forward="no">
  <topology_list>
    <topology type="physical" name="urn:publicid:IDN+ocf:i2cat">
      <node id="urn:publicid:IDN+openflow:ocf:i2cat:ofam+datapath+00:10::01" type="switch">
        <port num="10">
          <parameter type="status">
            <value timestamp="1409204400">1</value>
            <value timestamp="1409204410">1</value>
            <value timestamp="1409204420">2</value>
          </parameter>
          <parameter type="in-bps">
            <value timestamp="1409204400">100</value>
            <value timestamp="1409204410">110</value>
            <value timestamp="1409204420">100</value>
          </parameter>
          <parameter type="out-bps">
            <value timestamp="1409204400">200</value>
            <value timestamp="1409204410">210</value>
            <value timestamp="1409204420">200</value>
          </parameter>
        </port>
        <port num="11">
          <parameter type="status">
            <value timestamp="1409204400">2</value>
            <value timestamp="1409204410">2</value>
            <value timestamp="1409204420">1</value>
          </parameter>
          <parameter type="in-bps">
            <value timestamp="1409204400">100</value>
            <value timestamp="1409204410">110</value>
            <value timestamp="1409204420">100</value>
          </parameter>
        </port>
      </node>
      <node id="urn:publicid:IDN+openflow:ocf:i2cat:ofam+datapath+00:10::02"  type="switch">
        <port num="10">
          <parameter type="status">
            <value timestamp="1409204300">1</value>
            <value timestamp="1409204400">1</value>
            <value timestamp="1409204500">1</value>
          </parameter>
          <parameter type="in-bps">
            <value timestamp="1409204400">100</value>
            <value timestamp="1409204410">110</value>
            <value timestamp="1409204420">100</value>
          </parameter>
          <parameter type="out-bps">
            <value timestamp="1409204400">200</value>
            <value timestamp="1409204410">210</value>
            <value timestamp="1409204420">200</value>
          </parameter>
        </port>
      </node>
    </topology>
    <topology type="physical" name="urn:publicid:IDN+ocf:iMinds">
      ...
    </topology>
```

```
    </topology_list>
</monitoring-data>
```

3.1.2 Workflows

The MS interacts with other modules as in the workflow described in Figure 3.4 (this sequence diagram and any other appearing in this document are generated using www.websequencediagrams.com).

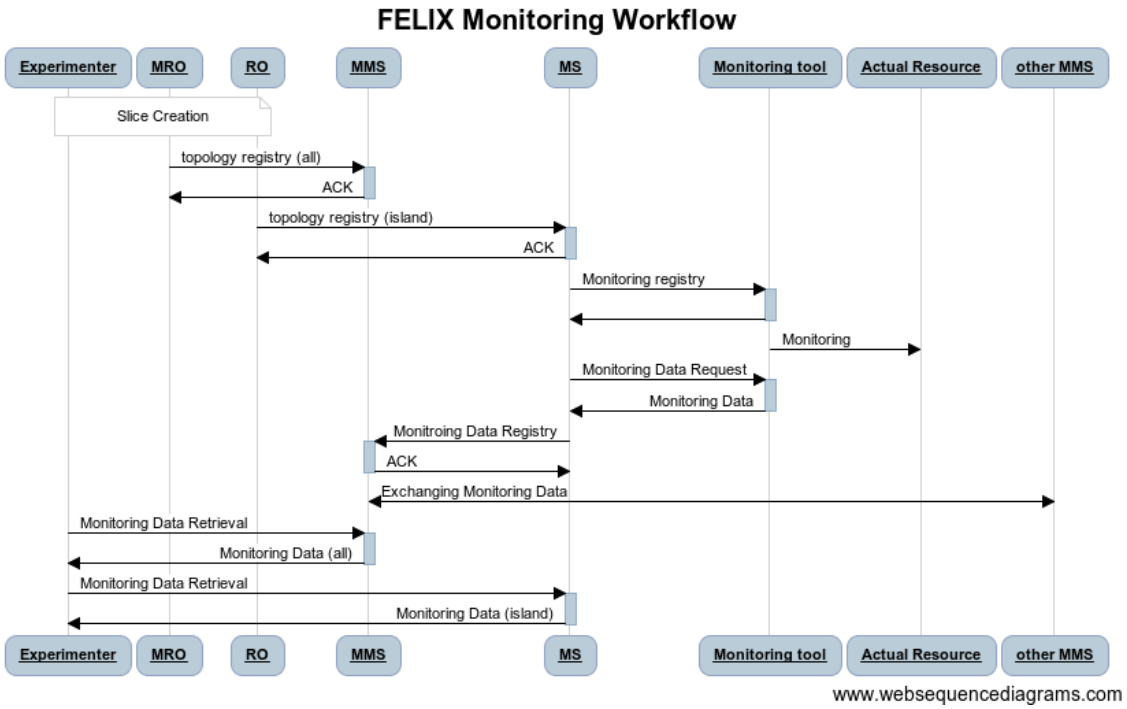

Figure 3.4: Monitoring Workflow

3.1.3 Future Work

Once the core functionality implementation of MS is finished, the following shall be addressed as future work:

- Finish the inter-communication process with other components such as RO, Expedient and individual monitoring tools.
- Consider interactions with newly developed RMs like TNRM and SERM.
- Integration of the AAA module with the MS.

3.2 SNMP Resource Monitor

Besides the MS, used to retrieve, aggregate and show monitoring metrics and other information; there is a complementary *SNMP Resource Monitor* tool that retrieves monitoring metrics directly from the resources available per slice. For more information on the design and internal operation of the *Simple Network Management Protocol* (SNMP), refer to the *Monitoring Tools* section.

3.2.1 Workflows

This monitoring tool directly contacts the resources (e.g. a virtual machine or a SDN-enabled switch) through the SNMP protocol and accessing them using a *pass phrase* (or *community name*) previously defined at the SNMP manager available per resource. After looking for a specific number of metrics on the desired resources, the retrieved data is persisted into an internal SQL database. This information can be aggregated and forwarded to another (M)MS module.

3.3 Monitoring Tools

This section contains the implementation details related to the resource monitoring tools (see Figure 3.1) that interact directly with the resources in order to collect data. These tools either allow to directly access the monitoring data (i.e. SNMP) or abstracts the access to the resources and presents it to the MS, for instance perfSONAR or Zabbix tools.

3.3.1 SNMP

The *Simple Network Management Protocol* (SNMP) is a popular protocol that has been standardised in a number of RFCs and whose objective is to manage network resources. This management process includes both 1) retrieving monitoring and configuration data, as well as 2) configuring network equipment directly. In the monitoring tool context, SNMPv2 is used for retrieving monitoring data from devices.

3.3.1.1 Design

SNMP is traditionally used for monitoring network equipment such as switches, but extensions also allow standardised representations of host resources (CPU load, memory, hard disk utilisation etc.). Such extensions are defined by MIBs (*Management Information Bases*), which describe the structure of management (and monitoring) data for a device, in a hierarchical namespace of object identifiers. For example, basic information about the device, such as its name, contact information etc. is available as in OID 1.3.6.1.2.1.1 "MIB-2 interfaces".

Information related to SDN resources can be retrieved from the MIB-2 interface table (1.3.6.1.2.1.2.2), status of computing resources can be found in the HOST-RESOURCES-MIB (1.3.6.1.2.1.25). See Appendix I for more information.

SNMP MIB-2 (SDN-RM)

SNMP MIB-2 [3] defines a structure containing information about every port (interface) belonging to a switch. This will be used to collect information on the bandwidth utilisation and on the status of networking resources from the SDN-RM.

Table I.1 in Appendix I gives an overview of the *ifEntry OID*. Some of the values in the *unit* column are further explained in RFC 1155 [4]. For the Ethernet interfaces, the values (as well as the *ifSpecific OID*) will adhere to the definitions of Managed Objects available at RFC 3635 [5]. As it can be observed from that table, SNMP exports the administration and operational statuses of an interface, the octets sent and received (typically since device power-up) and time stamp of the measurement ('ifLastChange').

SNMP HOST-RESOURCES-MIB (C-RM)

The HOST-RESOURCES-MIB is described in RFC 1514 [6]. This information can be found under OID 1.3.6.1.2.1.25 in the MIB hierarchy. This set of monitoring data can be retrieved on hosts, whether these are virtualisation servers or VMs in the slices. Therefore, this MIB could be used for monitoring any FELIX computing resource managed by C-RM.

The hierarchy contains information of a number of different resources. For example, *hrSystem* (OID 1.3.6.1.2.1.25.1) contains the entries denoted in Table I.2 from Appendix I. Another entry, the *hrStorage* (1.3.6.1.2.1.25.2), con-

tains information about storage devices. For example, *hrMemorySize* indicates the RAM or total physical read-write main memory (see Table I.3 from Appendix I).

Another example of data that can be retrieved uses the *hrStorageTable* object (1.3.6.1.2.1.25.2.3), which is the conceptual table of logical storage areas on the host. It contains multiple instances of *hrStorageEntry* describing storage, its size and available space. Refer to Table I.4 from Appendix I for further detail.

Another interesting object is *hrDevice* (1.3.6.1.2.1.25.3.3), which contains information on a number of devices, including a table *hrProcessorTable* for processors, consisting of *hrProcessorEntry* from which the CPU load can be extracted. Table I.5 in Appendix I provides additional information on this.

3.3.1.2 Workflows

The SNMP protocol is used to communicate between SNMP Managers (monitoring and aggregation) and SNMP Agents (on devices). The agent collects device status, monitoring data and counter *in situ* and offers it through the SNMP protocol. The MS contains a Monitoring Tool running an SNMP Manager that communicates with each of the agents of the set of resources monitored by the MS.

The Monitoring Tool will periodically request a subset of the SNMP MIBs from each resource, to later transform and store its monitored data into the Monitoring database, as pictured in Figure 3.4.

3.3.1.3 Future Work

While some existing proprietary MIBs are related to the SDN resources (here, OpenFlow), these are quite limited and are not supported by the SDN infrastructure provided by the domains in the FELIX Federation (that is, NEC switches and Open vSwitch devices).

For instance, an interesting metric in this context is the number of bytes sent/received for distinct OpenFlow flows; which allows monitoring SDN resources per slice. To retrieve such information we would need to use techniques specific to the resource. For example, Open vSwitch offers tools like *ovs-ofctl* and *ovs-dpctl* that allow retrieving some statistics on flows. However, this is not supported on the NEC hardware switches and this information is nevertheless not exposed through the SNMP protocol; so another monitoring tool would be necessary to fill this gap.

3.3.2 perfSONAR

Another monitoring tool is perfSONAR, a widely deployed test and measurement infrastructure that is used by some Research and Educational Networks such as GÉANT in Europe, JGN-X in Japan or Internet2 in the US. perfSONAR provides a uniform interface that allows scheduling measurements, storing data in uniform formats, and which provides scalable methods to retrieve data and generate visualisations.

3.3.2.1 Design

The perfSONAR measurement infrastructure is used by one of the network monitoring processes in the FELIX infrastructure. The perfSONAR software provides the monitoring data of network switches through the perfSONAR API, which is standardised in OGF. The *Monitoring Data Collector* component within MS collects the monitoring data through the perfSONAR API and utilises the data for FELIX slice monitoring.

3.3.2.2 Workflows

The current monitoring tool in the island monitors each of its switches and stores their monitoring data into the appropriate MS database. The *perfSONAR SequelService*, developed by NICT (*Japan's National Institute of Information and Communications Technology*), provides archived monitoring data and statuses through a perfSONAR interface in order to communicate with other monitoring tools. The *Monitoring Data Collector* of the MS collects the monitoring data accessing perfSONAR interface and imports such information to make it available within the FELIX Framework.

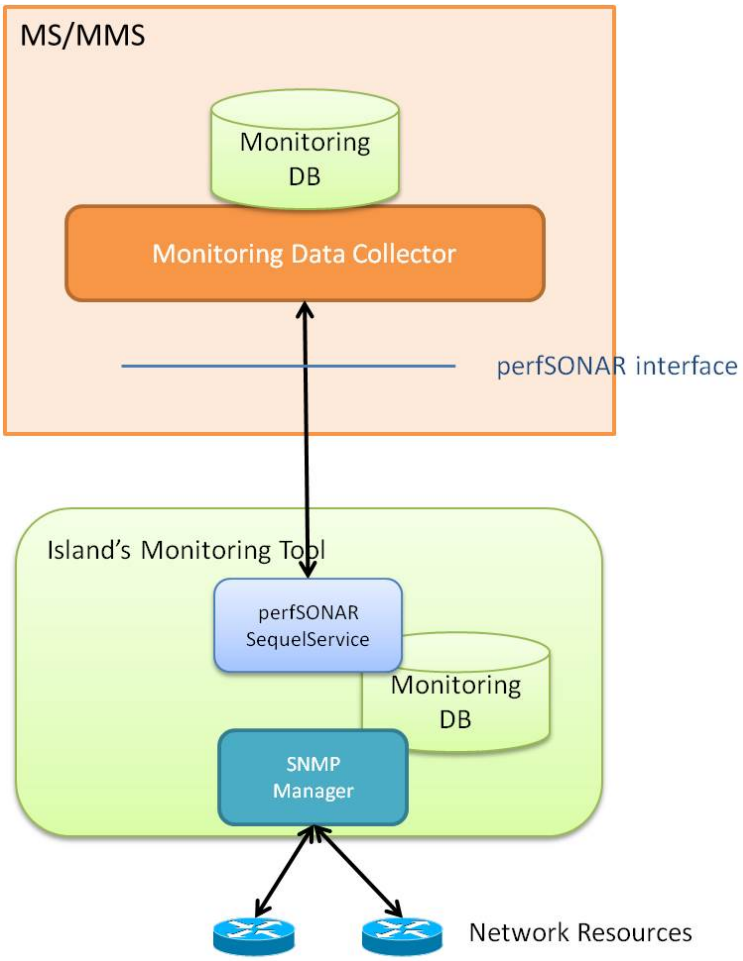

Figure 3.5: Design for perfSONAR

3.3.3 Zabbix

Zabbix is another open source software, commonly used for monitoring servers and networks. Its reports and statistics, as well as the configuration parameters, can be accessed through a web-based front-end. These can also be accessed through a specific API, so it can be easily accessed by other software components.

3.3.3.1 Design

Zabbix is used as one of the monitoring tools to monitor the computing resources in the FELIX Infrastructure. This tool provides monitoring and status data of physical servers and VMs through both an API and GUI. The *Monitoring Data Collector* component collects the monitoring data through an API exposed by Zabbix and later on aggregates that data to the FELIX computing resource monitoring.

3.3.3.2 Workflows

Zabbix is able to monitor the physical server, assuming a prior static configuration is performed. On the other hand, Zabbix monitors specific VMs, which are dynamically configured. The VMs are provisioned by an experimenter's request. At that very moment, some agent detects it and registers the VM information to Zabbix to start the monitoring. The *Monitoring Data Collector* component belonging to FELIX MS do collect the monitoring data through an API exposed by Zabbix API and makes the metrics of the VM available to the FELIX Framework, in a similar manner as before.

4 Deployment

This section provides information on how to deploy and the basic operations for both the Monitoring System (MS) module and the Public Monitoring. There is also some information on the Monitoring Tools used by the former modules.

4.1 Monitoring System

The MS module is composed of 2 main components: **MS API** and **Monitoring Data Collector**.

1. MS API
    - REST-based API that registers the topology information and monitoring to the database.
    - Provides such information stored in database to the experimenter through the GUI.
    - Written in Python.
2. Monitoring Data Collector
    - Collects periodically the monitoring data from islands' monitoring tools for all the resources and store them in the database.
    - Forwards the data to other MS modules if necessary.

4.1.1 Requirements and Dependencies

There is a small set of dependencies needed by the MS to work.

4.1.1.1 Requirements

The working environment must be a Debian-based distribution. Specifically, the MS is being developed under Debian 7 (Wheezy).

The MS requires the MySQL database because it is used for the topology information and monitoring data.

4.1.1.2 Dependencies

The packages required by MS are retrieved either by Debian's Advanced Packaging Tool (*apt-get*) and the Python version system (*easy_install*, *pip*).

Advanced packaging tool

The following Debian packages must be installed on the system for MS to properly work:

```
python apache2 mysql-server python-mysqldb python-elixir curl python-pip python-bottle
```

Pip

The following python packages must be installed on the system for MS to properly work:

```
xmltodict requests apscheduler
```

4.1.2 Configuration and Installation

4.1.2.1 Installation

Create the account to run the MS

```
# useradd mon_user
# visudo
mon_user ALL=(ALL) ALL
```

Save the *ms* branch under */opt/felix*:

```
login as mon_user
$mkdir -p /opt/felix/monitoring
git clone https://github.com/dana-i2cat/felix.git /opt/felix/monitoring
git checkout monitoring
```

Initializing the database.

```
$ mysql -u root < /opt/felix/monitoring/msjp/schema/monTopologyDB.sql
$ mysql -u root < /opt/felix/monitoring/msjp/schema/monSDNResourceDB.sql
```

4.1.2.2 Configuration

The */opt/felix/monitoring/conf* directory contains a set of configuration files that are used by the MS. These are explained as follows:

REST API parameters

The file *mon_api.conf* contains the parameters that configure the REST API of the MS module.

Section	Parameter	Type	Description
REST	rest_host	String	IP address of the server which provides MS API
REST	rest_port	Integer	Port number of MS REST API
URI	post_uri	String	URI of the MS that forwards the monitoring data
DATABASE	db_addr	Integer	IP address of the server that has the database to store the data
DATABASE	db_port	String	Port number of database service
DATABASE	db_user	String	user account to access the database
DATABASE	db_pass	String	password to access the database
LOG	log_dir	String	directory of the log file
LOG	log_file	String	log file name
UTILITY	debug	boolean	debug mode flag

Table 4.1: REST API configuration parameters

Monitoring Data Collector parameters

The file *mon_col.conf* contains the parameters of the Monitoring Data Collector component.

Section	Parameter	Type	Description
URI	post_uri	String	URI of the MS that forwards the monitoring data
URI	sequel_service_uri	String	URI of the monitoring tool for Japanese SDN infrastructure
URI	zabbix_uri	String	URI of the monitoring tool for Japanese Computing resources

DATABASE	db_addr	Integer	IP address of the server that has the database to store the data
DATABASE	db_port	String	Port number of database service
DATABASE	db_user	String	user account to access the database
DATABASE	db_pass	String	password to access the database
DATABASE	topology_db	String	name of the topology database
DATABASE	mon_data_sdn_db	String	name of the monitoring database for SDN
DATABASE	mon_data_cp_db	String	name of the monitoring database for Computing resources
DATABASE	mon_data_SE_db	String	name of the monitoring database for SE
DATABASE	mon_data_tn_db	String	name of the monitoring database for TN
MONITORING-ITEM	sdn_monitoring_item	boolean	monitoring metrics for SDN
MONITORING-ITEM	cp_monitoring_item_server	boolean	monitoring metrics for Computing hypervisor
MONITORING-ITEM	cp_monitoring_item_vm	boolean	monitoring metrics for computing resources (VMs)
MONITORING-ITEM	cp_monitoring_item_SE	boolean	monitoring metrics for SE
MONITORING-ITEM	cp_monitoring_item_TN	boolean	monitoring metrics for TN
LOG	log_dir	String	directory of log file
LOG	log_file	String	log file name
UTILITY	debug	boolean	debug mode flag

Table 4.2: Monitoring Data Collector configuration parameters

4.1.3 Operation

A typical set of operations performed on the MS comprises starting/stopping it and accessing through MS API.

REST API

Starting the server

```
$ /opt/felix/monitoring/msjp/monitoring_api.py
```

Stopping the server

```
type Ctrl+[c] or kill the process
```

Monitoring Data Collector

Starting the server

```
$ /opt/felix/monitoring/msjp/monitoring_data_collector.py
```

Stopping the server

```
type Ctrl+[c] or kill the process
```

4.2 SNMP Resource Monitor

This is a stand-alone application developed in Python implementing a REST API and SNMP manager. Monitoring data is persisted to an SQL database.

4.2.1 Requirements and Dependencies

The REST API (web-related) implementation uses the web.py library, which also provides database functionality. For the SNMP manager, the PySNMP library is used. For XML parsing, etree is used, which can be replaced by lxml if DTD validation is required. lxml is a binding of native code libxml2 and libxslt libraries.

To install these packages (on Debian):

```
apt-get install python-webpy python-pysnmp4 python-lxml
```

For the communication between MMS, it is recommended to provide the REST API via a public IP address.

4.2.2 Configuration and Installation

The *initdb.py* script is used to create a database and/or schema for the MS.

The *localsettings.py* file is used to configure the MS application. Its settings include:

- IP address and port for the REST API.
- Public IP address and port (in case the MS runs in a private address range and is forwarded through a firewall).
- IP address, port and url of a higher level MS or MMS where data is to be forwarded.
- Address, port and basic auth credentials (user:password) of the database.
- Community name for the SNMP manager.

4.2.3 Operation

Running the MS:

```
python ms.py
```

A summary log file will be created under */var/log*. The name of the file will depend on the PID of the MS application.

4.3 Monitoring Tools

In the following section we provide guidelines for configuring SNMP in both a specific NEC switch used by the FELIX Infrastructure and an instance of Open VSwitch, as well as instructions on deploying and testing the SNMP package on a Debian server.

4.3.1 SNMP

4.3.1.1 Requirements and Dependencies

For switches, we rely on the availability of *SNMP MIB-2* (OID .1.3.6.1.2.1.2). As for the servers (computing resources), we rely on the *HOST-RESOURCES-MIB* (.1.3.6.1.2.1.25).

A hardware switch should already have MIB-2 available. Instead, for software switches (Open vSwitch) and servers (Linux, Debian); an SNMP daemon should be installed.

4.3.1.2 Configuration and Installation

This section explains the configuration of SNMP for hardware switches, and the installation and configuration for Open vSwitch and host resources.

To test SNMP on a device, use:

```
snmpwalk -v2c -c &lt;community_name&gt; &lt;switch_IP&gt;
```

The *snmpwalk* command is available on the Debian *snmp* package.

<community_name> is the name of the SNMP community, which is previously chosen by the switch administrator. Default community names are typically public or private. Community names are local to island and not passed through the monitoring framework. Likewise, access to the SNMP agent is local to island and not available to other FELIX partners (i.e., firewalled or on a different network).

When using persistent monitoring of any kind (including SNMP) as well as creating slices on Open vSwitch, it is important for consistency to make sure that the network ports (e.g., tap, eth1, veth etc.) added to a switch are always assigned the same OpenFlow port number. **Older versions of Open vSwitch do not allow to fix an OpenFlow port number for interfaces** added to a switch. This means that upon rebooting the machine, interfaces may be assigned random port numbers, which is problematic since flowspaces are constructed using these port numbers (not interface names), and SNMP reports port names, not OpenFlow port numbers. At least version 1.10.0-1 of Open vSwitch must be used.

To add a port with a certain fixed OF port number (e.g., interface *tap2* as OF port number *2* on switch *myswitch*):

```
ovs-vsctl add-port myswitch tap2 -- set Interface tap2 ofport_request=2
```

The ofport_request is kept in a separate column in the Open vSwitch database and is persistent (unlike the ofport column). Note that the add-port and set command must be on the same line to be effective.

NEC 3640-xx/8800

You may want to set the hostname of the switch so it appears as sysName in SNMP MIB.

SNMP is always running. To enable access for an SNMP manager to retrieve MIB from the agent on the switch, create a SNMP community (in enable, config context).

```
# snmp-server community <community_name> ro
# save
```

To restrict access not just by community, but also by IP address (of the SNMP manager), a generic access-list (with ID=**1** in the example below) can be created, and then the community is restricted to this list:

```
# access-list 1 permit 10.216.132.12 0.0.0.0
# snmp-server community <community_name> ro <1>
# save
```

Open vSwitch

Open vSwitch does not support SNMP. Instead, it uses the Linux SNMP [7] daemon to run an agent to collect port counters.

1. Install net-snmp
    - apt-get install snmpd
2. Configure the SNMP agent using the */etc/snmp/snmpd.conf* configuration file
    - set the listen address(es) for the SNMP agent.
    - Listen for connections from management interface
    - agentAddress udp:10.216.132.5:161
3. create a view which restricts the reported MIBs:

- `view interfaces included .1.3.6.1.2.1.2`
- The default snmpd.conf has examples under ACCESS CONTROL. It has a system-only view which allows access to system information (.1.3.6.1.2.1.1) and host resources (.1.3.6.1.2.1.25.1) (i.e., CPU, processes, disks).

4. create a community which exposes this view (second parameter, here 'default', is the IP address access filter for the community):
   - `rocommunity <community_name> default -V interfaces`
5. save the file and restart snmpd
   - `service snmpd restart`

Computing resources

Since SNMP for computing resources is also based on an SNMP daemon, the installation and configuration is similar to that of SNMP for Open vSwitch. Instead, add a `host` view in the */etc/snmp/snmpd.conf* configuration file that exposes the HOST-RESOURCES-MIB:

```
view   host        included   .1.3.6.1.2.1.25
```

4.3.1.3 Operation

After the configuration, monitoring data will be automatically collected by the devices and become available through the SNMP API while the device (that is, a switch or server) is running.

5 Conclusions and Summary

In this document we have addressed the two most important ways in which the FELIX Framework retrieves data relevant to the monitoring process. We detailed the design, implementation and operation of the MS module and we also introduced the SNMP Resource Monitor along with the Monitoring Tools used directly or indirectly from the two aforementioned modules. At the end of the deliverable we also present brief guides for the deployment of the FELIX modules and some of the tools.

The MS module consists of a number of components that work closely in order to retrieve, aggregate and interpret monitoring data. Besides, it also receives and interprets information from the RO that contains both physical and slice topologies. Obtaining the final data is carried out in several manners: from direct access to resources to inter-communication with other FELIX modules. In the end, this information is aggregated in the upper layers so as to have an overview of the whole network and to identify the usage of the resources provisioned within the active slices.

We conclude that the monitoring information is an important element when seeking an accurate decision making process. Such process is to take place in the upper layers only after data has been comprehensively fetched and aggregated from the underlying layers. Future work shall focus on finishing the integration of the MS with other related components.

References

[1] ``perfSONAR.'' https://www.perfsonar.net.

[2] ``Zabbix.'' http://www.zabbix.com.

[3] ``Management Information Base for Network Management of TCP/IP-based internets: MIB-II,'' IETF RFC 1213, 1991.

[4] ``Structure and Identification of Management Information for TCP/IP-based Internets,'' IETF RFC 1155, May 1991.

[5] ``Definitions of Managed Objects for the Ethernet-like Interface Types,'' IETF RFC 3635, Sept. 2003.

[6] ``Host Resources MIB,'' IETF RFC 1514, Sept. 1993.

[7] ``Net-SNMP tool information.'' http://www.net-snmp.org/.

Appendix I

In this appendix we present some complementary information on the objects available through the SNMP protocol that were previously introduced in the *Implementation* section.

A SNMP metrics for SDN-RM resources

OID	Description	Type	Unit	Mandatory
1.3.6.1.2.1.2.2.1.1	ifIndex	Integer	--	x
1.3.6.1.2.1.2.2.1.2	ifDescr (interface name)	String	--	x
1.3.6.1.2.1.2.2.1.3	ifType	Integer	enum (see [3])	x
1.3.6.1.2.1.2.2.1.4	ifMtu (interface MTU)	Integer	Octets	x
1.3.6.1.2.1.2.2.1.5	ifSpeed	Gauge (bounded non-negative Integer)	bits/s	x
1.3.6.1.2.1.2.2.1.6	ifPhysAddress	PhysAddress	--	x
1.3.6.1.2.1.2.2.1.7	ifAdminStatus	Integer	up / down / testing	x
1.3.6.1.2.1.2.2.1.8	ifOperStatus	Integer	up / down / testing	x
1.3.6.1.2.1.2.2.1.9	ifLastChange (time stamp)	TimeTicks	0.01s	x
1.3.6.1.2.1.2.2.1.10	ifInOctets	Counter	Octets	x
1.3.6.1.2.1.2.2.1.11	ifInUcastPkts	Counter	Packets	x
1.3.6.1.2.1.2.2.1.12	ifInNUcastPkts	Counter	Packets	x
1.3.6.1.2.1.2.2.1.13	ifInDiscards	Counter	Packets	x
1.3.6.1.2.1.2.2.1.14	ifInErrors	Counter	Packets	x
1.3.6.1.2.1.2.2.1.15	ifInUnknownProtos	Counter	Packets	x
1.3.6.1.2.1.2.2.1.16	ifOutOctets	Counter	Octets	x
1.3.6.1.2.1.2.2.1.17	ifOutUcastPkts	Counter	Packets	x
1.3.6.1.2.1.2.2.1.18	ifOutNUcastPkts	Counter	Packets	x
1.3.6.1.2.1.2.2.1.19	ifOutDiscards	Counter	Packets	x
1.3.6.1.2.1.2.2.1.20	ifOutErrors	Counter	Packets	x
1.3.6.1.2.1.2.2.1.21	ifOutQLen	Gauge	Packets	x
1.3.6.1.2.1.2.2.1.22	ifSpecific	OID	--	x

Table I.1: Overview of the ifEntry object

B SNMP metrics for C-RM resources

OID	Description	Type	Unit	Mandatory
1.3.6.1.2.1.25.1.1	hrSystemUptime	TimeTicks	0.01s	x
1.3.6.1.2.1.25.1.2	hrSystemDate	DateAndTime (String)	--	x
1.3.6.1.2.1.25.1.3	hrSystemInitialLoadDevice	Integer	--	x

1.3.6.1.2.1.25.1.4	hrSystemInitialLoadParameters	InternationDisplay String	--	x
1.3.6.1.2.1.25.1.5	hrSystemNumUsers	Gauge (bounded non-negative Integer)	user sessions	x
1.3.6.1.2.1.25.1.6	hrSystemProcesses	Gauge	process con-texts	x
1.3.6.1.2.1.25.1.7	hrSystemMaxProcesses	Integer	process con-texts	x

Table I.2: Overview of the hrSystem object

OID	Description	Type	Unit	Mandatory
1.3.6.1.2.1.25.2.2	hrMemorySize	KBytes	1024 bytes	x

Table I.3: Overview of the hrMemorySize object

OID	Description	Type	Unit	Mandatory
1.3.6.1.2.1.25.3.1.1	hrStorageIndex	Integer	--	x
1.3.6.1.2.1.25.3.1.2	hrStorageType	OID	--	x
1.3.6.1.2.1.25.3.1.3	hrStorageDescr	String	--	x
1.3.6.1.2.1.25.3.1.4	hrStorageAllocationUnits	Integer	bytes	x
1.3.6.1.2.1.25.3.1.5	hrStorageSize	Integer	hrStorageAllocationUnits bytes	x
1.3.6.1.2.1.25.3.1.6	hrStorageUsed	Integer	hrStorageAllocationUnits bytes	x
1.3.6.1.2.1.25.3.1.7	hrStorageAllocationFailures	Counter	--	x

Table I.4: Overview of the hrStorageEntry object

OID	Description	Type	Unit	Mandatory
1.3.6.1.2.1.25.3.3.1.1	hrProcessorFrwID	ProductID (OID)	--	x
1.3.6.1.2.1.25.3.3.1.2	hrProcessorLoad	Integer (0..100)	percentage	x

Table I.5: Overview of the hrStorageEntry object