Project Acronym	**Fed4FIRE**
Project Title	**Federation for FIRE**
Instrument	**Large scale integrating project (IP)**
Call identifier	**FP7-ICT-2011-8**
Project number	**318389**
Project website	**www.fed4fire.eu**

D6.6 – Report on third cycle development regarding measuring and monitoring

Work package	WP6
Task	T6.1, T6.2, T6.3
Due date	29/02/2016
Submission date	03/02/2016
Deliverable lead	Yahya Al-Hazmi (TUB), Alexander Willner (TUB)
Version	V2.0
Authors	Yahya Al-Hazmi (TUB) Alexander Willner (TUB) Tim Wauters (iMinds) Olivier Mehani (NICTA) Thierry Rakotoarivelo (NICTA) Donatos Stavropoulos (UTH) Loïc Baron (UPMC)
Reviewers	Mikhail Smirnov (FOKUS) and Carlos Bermudo (i2CAT)

Abstract	This deliverable reports on the developments of the third cycle regarding monitoring and measurements across the Fed4FIRE federation and used monitoring tools.
Keywords	Measurement, Monitoring, OML, Semantic, Ontologies, Security, Manifold, SLA, Reputation, Reservation.

Nature of the deliverable	R	Report	X
	P	Prototype	
	D	Demonstrator	
	O	Other	
Dissemination level	PU	Public	X
	PP	Restricted to other programme participants (including the Commission)	
	RE	Restricted to a group specified by the consortium (including the Commission)	
	CO	Confidential, only for members of the consortium (including the Commission)	

Disclaimer#

The information, documentation and figures available in this deliverable, is written by the Fed4FIRE (Federation for FIRE) – *project consortium and does not necessarily reflect the views of the European Commission. The European Commission is not liable for any use that may be made of the information contained herein.*

The Fed4FIRE project received funding from the European Union's Seventh Framework Programme for research, technological development and demonstration under grant agreement no FP7-ICT-318389.

Executive Summary

This deliverable reports on the third cycle development of the Fed4FIRE monitoring and measurement architecture covered by the work package 6. The first and second cycle development focused on the implementation of the three main types: facility monitoring, infrastructure monitoring and experiment measuring. The third cycle focused on the significant add-on features implemented to complete the monitoring and measurement architecture as well as fulfilling requirements from other Fed4FIRE work packages and communities. This includes the introduction of a common information model for monitoring information, the storage of this information in a semantic database, work on the secure transport of measurement information, and easier access to the monitoring data.

Acronyms and Abbreviations

API	Application Programming Interface
AM	Aggregate Manager
DB	Database
FLS	First Level Support
GENI	Global Environment for Network Innovations
HTTP	Hypertext Transfer Protocol
IP	Internet Protocol
ICMP	Internet Control Message Protocol
FIRE	Future Internet Research and Experimentation
FLS	First Level Support
NIC	Network Interface Controller
OML	ORBIT Measurement Library
OMN	Open-Multinet
PSK	Pre-Shared Key
RDF	Resource Description Framework
RDFS	Resource Description Language Schema
SPARQL	SPARQL Protocol and RDF Query Language
SQL	Structured Query Language
SSH	Secure Shell
TLS	Transport Level Security

Table of Contents

List of Figures

1 Introduction

This deliverable reports on the third cycle development of the monitoring and measurement architecture that was described in the D6.4 “Detailed specifications regarding monitoring and measurement for third cycle ready” [1]. These developments include a common information model based on semantics and the respective extensions in OML in order to demonstrate a proof of concept. Moreover, a new version of OML addresses security limitations described in the D6.4 by introducing a mechanism for proper identity management and access control. Regarding the user-friendliness of data access and visualization two mechanisms have been developed with the first supporting SPARQL queries, while the second using Manifold for accessing the data. The rest of the document is structured as follows. In Section 2 requirements derived from the “Federation Architecture”, the main stakeholders and WP6 itself, are presented. Section 3 describes the implementation details of several main architectural elements. These include the common information model, extensions to OML regarding semantic support and new security features and mechanisms for data retrieval and visualization. Finally, a conclusion of the deliverable is given in Section 4.

2 Input to this deliverable

This section gives a brief summary on requirements from different stakeholders in Fed4FIRE that are relevant to monitoring development in the third cycle.

2.1 Architecture

With regards to monitoring, no architectural changes have been proposed for the existing monitoring components in Deliverable D2.7 "Third Federation Architecture" [4], compared to previous cycles. In the architecture three types of monitoring are identified: facility monitoring, infrastructure monitoring and experiment measuring (Figure 1 and Figure 2).
This deliverable focuses on the facility and infrastructure monitoring services that are implemented in the third cycle. These services are related to the monitoring of the availability and of the health status of testbeds involved in the Fed4FIRE federation and of the infrastructure resources by the testbed providers, respectively.
For the testbed side, they are defined by the architecture (D2.7) as follows:

- **"Facility monitoring**: this monitoring is used in the first level support to see if the testbeds are still up and running. The testbed has the freedom to adopt any solution to gather this type of monitoring data as it sees fit (e.g. an existing monitoring framework such as Zabbix, Nagios or similar), as long as it is able to export that data as an OML stream to the Federator's central OML server, which will store it in a database for First Level Support. In the first cycle of Fed4FIRE, the facility monitoring was rolled out on all testbeds."
- **"Infrastructure monitoring:** instrumentation of resources by the testbed provider itself to collect data on the behavior and performance of services, technologies, and protocols. This allows the experimenter to obtain monitoring information about the used resources that he could not collect himself. Examples of such infrastructure monitoring data are information regarding the CPU load and NIC congestion on the physical host of a virtual machine resource, the monitoring data of switch traffic, or the gathering of data regarding the wireless spectrum during the course of the experiment."

Infrastructure monitoring data can be provided for federation services (trustworthy reputation service, reservation broker, SLA management), as well as for experimenters (e.g. via pushing specific resource monitoring data). At the federator side, distinction is made between the following components:

- The **FLS dashboard** gives a real-time, comprehensive but also very compact overview of the health status of the different testbeds included in the Fed4FIRE federation. To determine this health status, it combines facility monitoring information provided by the testbeds with specific measurements performed by the dashboard component itself.
- The federator (iMinds in this case) provides an **OML server and corresponding database for FLS data** to process and store facility monitoring data of the testbeds to be used by the FLS.
- The **data broker** is an optional component that can be **accessed through the portal**, and which makes it easier for some federation services (like reputation service and reservation broker) as well as novice experimenters to retrieve their experiment data from the different sources where it might reside (OML servers of the different testbeds that provided infrastructure monitoring, OML servers of the experimenter itself on which the experiment measurements were stored, etc.).

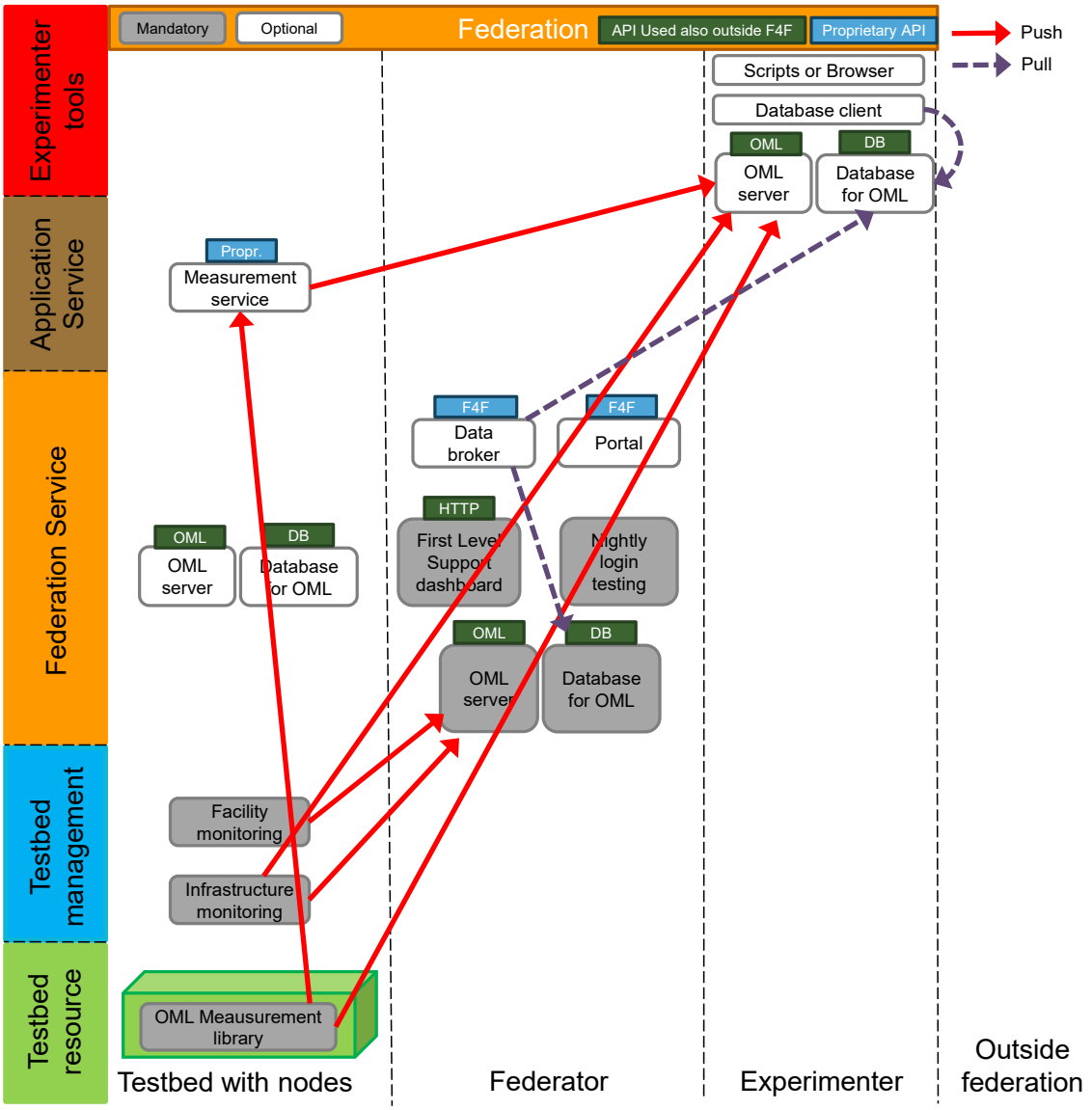

Figure 1: Monitoring and measurement architecture for cycle 3.

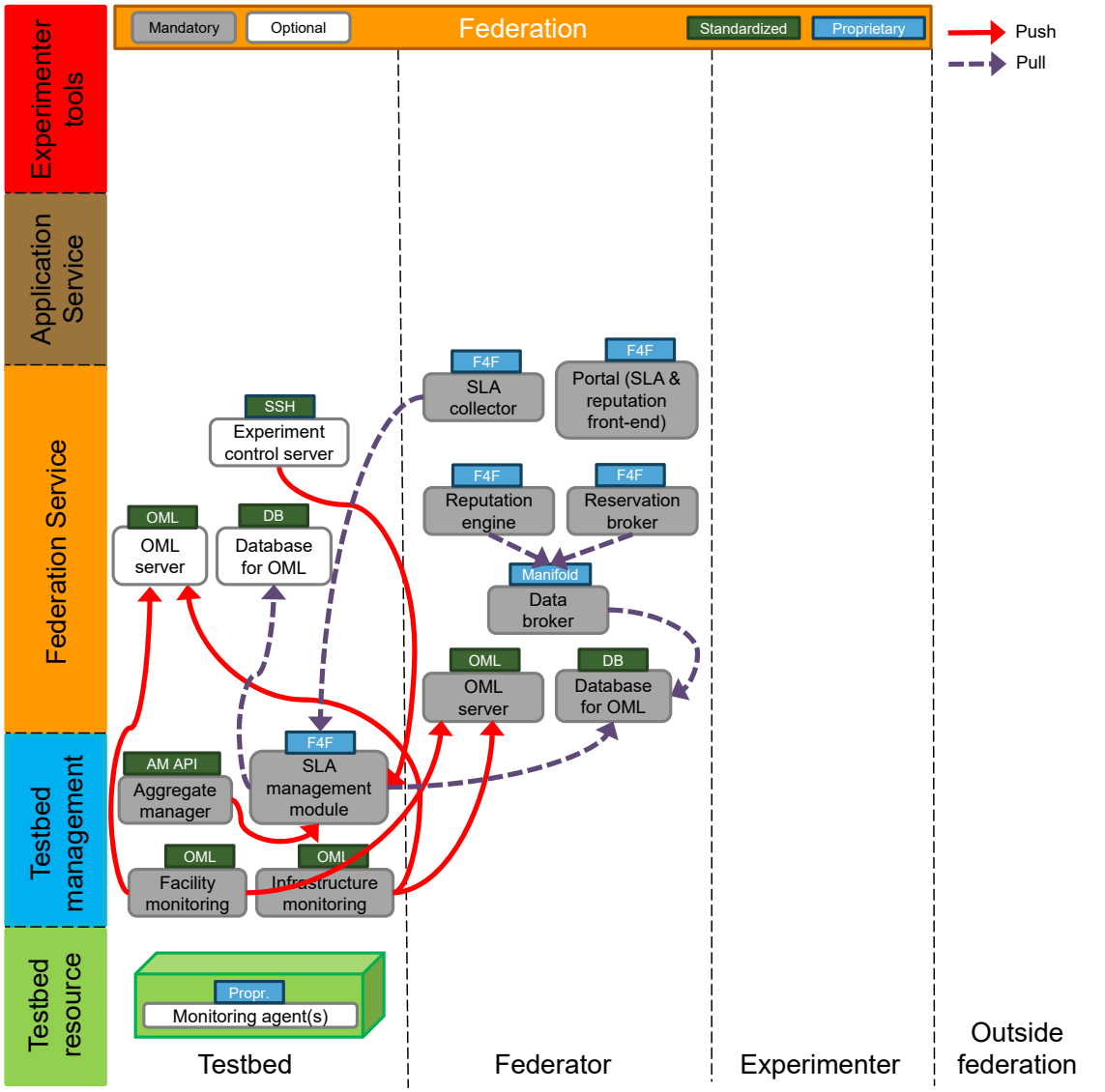

Figure 2: Relation between the monitoring and measurement architecture and the SLA, reputation and future reservation mechanisms as defined in cycle 3.

2.2 High priority requirements from the main stakeholders

This section recalls the requirements relevant to the third cycle development of WP6. Deliverable D8.7 [5] deals with additional implications for facility monitoring to provide an optimal FLS service. It is mainly concerned with the requirement to operate in a federated environment, which necessarily leads to a new obligation on a testbed for the active provision of operational monitoring information to FLS.

After the second cycle of operation, additional observations have been identified, with regards to monitoring. These result in requirements related to FLS process automation, enhanced testing and reporting. These requirements have been implemented in order to provide a more comprehensive FLS

monitoring dashboard: additional information is now available on the aggregated status and the login status of testbeds, and automated e-mail distribution is in place.

In comparison to the second cycle, no additional requirements have been defined for facility or infrastructure monitoring by experimenters, in the combined WP3 and WP4 deliverable D3.4/D4.4 [6].

2.3 Additional WP6 requirements

In addition to the general requirements from the main stakeholders discussed in the previous section, three additional features have been identified by WP6 in the third cycle to enhance the monitoring and measurement architecture:

- The use of common monitoring and measurement information model in order to represent the data in unified and meaningful way (described in Sections 3.1 and 3.2),
- Providing a secure data collection and access control (described in Section 3.3), and
- Providing user data in a user- friendly manner (described in Section 3.4).

2.4 Deviations from specifications in D6.4

In the third development cycle we have met those requirements from the architecture as well as other stakeholders concerning the infrastructure monitoring as well as the specifications defined for the second cycle. Thus, there are no deviations from the D6.4 [1].

3 Implementation of the main architectural elements

3.1 Common Information Model

This section discusses the implementation of the information model within the Fed4FIRE monitoring and measurement architecture following the design discussed in D6.4 [1].

As discussed in D6.4 [1], an ontology-based information model is designed that models all monitoring and measurement related concepts and their relationships. It also models those components, technologies and protocols used in the Fed4FIRE monitoring architecture. This model is part of a larger model that models federated infrastructures with focus in supporting the entire experiment lifecycle. This model is called Open-Multinet (OMN) Ontology [7]. Its development is lead by Task 5.2 in WP5; however, this activity is performed across technical WPs (WP5, WP6 and WP7).

OMN ontology comprises a set of ontologies focusing on modelling different aspects of testbeds federation such as infrastructures, resources, services, lifecycle, monitoring, policies, etc. A set of sub-ontologies are dealing with different monitoring aspects: an upper ontology that describes the basic concepts and relations, in addition to other five sub-ontologies modelling measurement metrics, data, units, tools, and generic concepts.

OMN ontologies are implemented within the monitoring architecture. As OML [3] is used as the main tool in the architecture, a new version of OML was released that supports the semantics, the use of OMN ontologies, and the Resource Description Framework (RDF) [8], which uses the RDF Schema (RDFS) [9] as a data model.

3.2 Semantic OML

This section discusses the implementation of the new OML release, called semantic OML. Figure 3 shows a high level overview on the implementation as discussed also in D6.4. OML has been extended at the client and server sides in order to adopt OMN ontologies and support representing and storing the data as RDF triples.

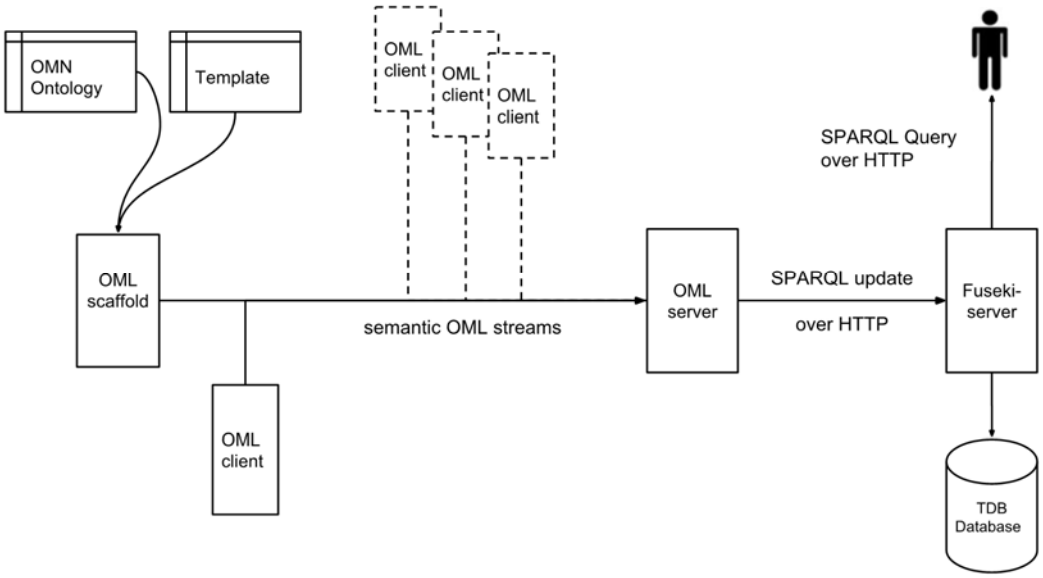

Figure 3: Semantic OML

3.2.1 OML extension at the client

As shown in Figure 3, the OML scaffold (oml2-scaffold) is used to generate skeleton OML application source code (used for injecting monitoring data to be transported to the server) based on a template (e.g. shown in Listing 1) provided by the user. This code is part of the OML client. OML scaffold is extended to support the creation of the injection code following the developed ontology in order to build the insert statements semantically.

It supports the creation of the code in three different languages; besides the natively supported C, Python and Ruby are supported. As input, scaffold gets the OMN ontologies as well as a user template that includes the target RDF-based schemas to represent the required measurement metrics and their related resources about which monitoring data are provided. For instance, the schema represented in Listing 1 is used to provide infrastructure-monitoring service; it includes only schemata for representing the used memory measurement metric of a physical server (PC) that hosts a user VM, using the OMN ontology as the information model. Scaffold is also extended to validate the correct use of RDF schemas in the template against OMN ontologies.

In this implementation, it is decided that the semantic schema has to be created following some rules, these are:

1. Every semantic schema has to begin with *SimpleMeasurement* (as the subject in the first triple), followed by its corresponding Metric class (as object) *omn-monitoring-metric:<MetricName>.*
   Example: omn-monitoring-data:SimpleMeasurement omn-monitoring:isMeasurementDataOf omn-monitoring-metric:UsedMemory
2. The metric has to be linked with a resource by the property *omn-monitoring:isMeasurementMetricOf.*
   Example: omn-monitoring-metric:UsedMemory omn-monitoring:isMeasurementMetricOf omn-domain-pc:PC

3. Later on, a URI of that resource must be given
   Example: omn-domain-pc:PC omn:hasURI %value%

Note that the prefixes of all ontologies used in the schemas (in triples) have to be predefined to the oml2-scaffold.

```
defApplication('oml:infrastructure-monitoring', 'infrastructure_monitoring') do
|app|
        app.version(1,0)
        app.shortDescription = "Monitoring"
        app.description = %{an application to provide infrastructure resource
monitoring}

app.defMeasurement("used_memory"){ |m|
    m.defMetric('used_memory', :double, 'Used memory value of host',
    [['omn-monitoring-data:SimpleMeasurement','omn-monitoring-
data:isMeasurementDataOf','omn-monitoring-metric:UsedMemory'],
    ['omn-monitoring-metric:UsedMemory','omn-
monitoring:isMeasurementMetricOf','omn-domain-pc:PC'],
    ['omn-monitoring-data:SimpleMeasurement','omn-monitoring-
data:hasMeasurementDataValue','%value%'],
    ['omn-monitoring-data:SimpleMeasurement','omn-monitoring:hasUnit','omn-
monitoring-unit:Byte'],
    ['omn-monitoring-unit:Byte','omn-monitoring-unit:hasPrefix','omn-monitoring-
unit:giga']])

    m.defMetric('timestamp', :datetime, 'Time when the metric is measured',
    [['omn-monitoring-data:SimpleMeasurement','omn-monitoring-
data:hasTimestamp','%value%']])

    m.defMetric('physicalresource', :string, 'URI of monitored resource',
    [['omn-domain-pc:PC','omn:hasURI','%value%']])

    m.defMetric('virtualresource', :string, 'URI of virtual host which is running
on the monitored physical host',
    [['omn-domain-pc:VM','omn:hasURI','%value%'],
    ['omn-domain-pc:VM','omn-lifecycle:childOf','omn-domain-pc:PC']])}
```

Listing 1: RDF-based monitoring schema template

As aforementioned, the code generated by scaffold is part of the OML client that retrieves then the real data from monitoring tools deployed at the testbed. Listing 2 shows the code generated by the Scaffold for the schema represented in Listing 1.

```
import oml4py
omlInst = oml4py.OMLBase(<app-name>, <domain-name>, <sender-ID>, <OML-server-URI>)
#--------------------------------------------------------------------------# □
omlInst.addmp("used_memory", "used_memory:double:{omn-monitoring-
data:SimpleMeasurement|omn-monitoring-data:isMeasurementDataOf|omn-monitoring-
metric:UsedMemory}{omn-monitoring-metric:UsedMemory|omn-
monitoring:isMeasurementMetricOf|omn-domain-pc:PC}{omn-monitoring-
data:SimpleMeasurement|omn-monitoring-data:hasMeasurementDataValue|%value%}{omn-
monitoring-data:SimpleMeasurement|omn-monitoring:hasUnit|omn-monitoring-
unit:Byte}{omn-monitoring-unit:Byte|omn-monitoring-unit:hasPrefix|omn-monitoring-
unit:giga} timestamp:datetime:{omn-monitoring-data:SimpleMeasurement|omn-
```

```
monitoring-data:hasTimestamp|%value%} physicalresource:string:{omn-domain-
pc:PC|omn:hasURI|%value%} virtualresource:string:{omn-domain-
pc:VM|omn:hasURI|%value%}{omn-domain-pc:VM|omn-lifecycle:childOf|omn-domain-pc:PC}
")
#------------------------------------------------------------------------# □
oml_register_mps() □
omlInst.start() □
#write the wrapper code here to retrieve the real data values from local tools
omlInst.inject(<mpname>, <values>) □
omlInst.close()
```

Listing 2: RDF-based injection point generated by OML Scaffold in Python to be used as part of an OML wrapper

The placeholders of the schema are explained as follows:

- *app-name*: name of application or wrapper
- *domain-name*: name of infrastructure the experiment is executed in
- *sender-ID*: ID of the sender, usually the IP address
- *OML-server-URI*: IP address and port number of the OML server.

The new version of OML Scaffold that supports the extension is provided at [10]. Examples on the OML wrappers written in C, Python and Ruby are provided as well in [10] along with examples of some templates.

How to use the OML Scaffold?

The oml2-scaffold file can be deployed on any Linux machine on which you want it be. In case the semantic OML server is deployed, this file is already deployed either under /usr/local/bin/, or in the source code directory of the server under soml/ruby/.

To enable semantic validating, install 'rdf/rdfxml' and 'text' from ruby gem, for example using the following commands:

```
gem install rdf-rdfxml -v 1.1.3
gem install text -v 1.3.0
```

Now you can use oml2-scaffold for semantic validation with the following command:

```
/path/to/oml2-scaffold --oml <path-to-schema.rb> --ontology <ontology-dir>
```

To generate code in Python file, use `type --omlpy` instead of `--oml`.
To generate code in Ruby file, use type `--omlr` instead of `--oml`.

Regarding C, scaffold will generate a skeleton C-header file, which serves as a framework for the OML client / wrapper. The header will be compiled together with the main C-file to generate an executable C-program that can send measurement streams to OML Server. The main file can be written by the user or the scaffold can be used to create a template for the main C-file, which needs to be modified

to suite the needs (by template, it means that the measurement data are generated randomly, this serves as testing purposes because you can immediately test it without having to add any data at all).

The following commands are important if C is used:
To create the header file, use:

```
/path/to/oml2-scaffold --oml <path-to-schema.rb> --ontology <ontology-dir>
```

To create the main file, use:

```
/path/to/oml2-scaffold --main <path-to-schema.rb> --ontology <ontology-dir>
```

To enable command-line parsing, use:

```
/path/to/oml2-scaffold --opts <path-to-schema.rb>
```

To compile:

```
cc -c -Wall -Werror -g -I.  generator.c -o generator.o
gcc -o <filename> <filename>.o -loml2 -lpopt
cc -o filename filename.o  -loml2 -lpopt
```

After executing these commands an executable file is generated that is used to send measurement data as OML following RDF-based schemas using the following command:

```
/path/to/<client-program> --oml-id <id> --oml-domain <domain> --oml-collect
tcp:<server IP>:<port>
```

In a case that the data are queried from Zabbix (which doesn't have C API), Python file will be used to query the data from Zabbix. Within this file the executable C-program is called with the data given in parameters. Example:

```
/path/to/<client-program> --oml-id <id> --oml-domain <domain> --oml-collect
tcp:<server IP>:<port> --used-bandwidth <value> --time <timestamp> --pm-used-
bandwidth <monitored physical host> --vm-used-bandwidth <monitored virtual
host>
```

The parameters' names (e.g. --used-bandwidth, --time) must be defined beforehand in the schema.

In the current use of OML in Fed4FIRE OML clients / wrappers, testbed providers need to write OML clients/wrappers to provide the data. Minimum efforts are required by a testbed provider to support the semantics and the use of the ontology information model. All what a testbed provider needs to do is to replace the code used in the currently running OML clients / wrappers with the code generated by the Scaffold. This new generated code caters for injecting the data following RDF-based schemas that are also defined by the testbed provider as well.
To be noticed that it is not necessary to use Scaffold to generate such code, if a code is available but some modifications to an existing code is used. However, using Scaffold ensures the correct use of the terms and vocabularies in the used schemas.
OML libraries (oml2lib, OML4R) can be further used, but OML4Py needs to be replaced with the one at [10] as the original version does not support flexible schemas.

To use Scaffold, Ruby v1.9.2+ is required, and install the necessary extensions RDF-RDFXML[1] and the text[2] tool.

3.2.2 OML extension at the server

The implemented semantic OML server is available at [10]. It supports semantics, understands RDF-based schemas written following the OMN ontologies and can parse and process the data represented in the user-defined template discussed in the previous section. The server inserts the data into Fuseki server [12] that stores the data in Jena TDB triple store using the SPARQL Update over HTTP.

To replace an already deployed OML server, it is only needed to replace the old version with the new one at [10] and recompile.
If no server is already deployed, download the new source code from [10] and build the server following the instructions provided in [11].

Then install Jena Apache Fuseki server with TDB following the instruction given in [12].

To start Fuseki server:

Use the following command to start Fuseki
`/path/to/fuseki-server --update --loc=<location> /<dataset>`

To use in-memory dataset:
`/path/to/fuseki-server --update --mem=/<dataset>`

`--update` option allows SPARQL Update, whereas SPARQL Query is enabled by default
`--loc` allows to choose the location (directory) where the database should be stored
`--/<dataset>` identifies the path to the dataset
`--mem` creates an in-memory dataset in case `--loc` is not used

Note that for the creation of multiple databases for different users (exporters) a configuration file is required; see Fuseki documentation for more details.

Start semantic OML-Server:

```
/path/to/oml2-server -b fuseki --fus-namespace <dataset> --fus-host <host> -
-fus-port <port>
```

`<dataset>` is the same as the one identified in fuseki-server
Note that the native database backend (posgresql or sqlite) can still be used by using the option -b ("-b posgresql" or "-b sqlite")

[1] https://rubygems.org/gems/rdf-rdfxml

[2] https://rubygems.org/gems/text

3.3 Secure OML

Work is underway for dependencies of the feature, but the Secure OML transport is not currently functional yet. Support for this depends on three functionalities.

1. TLS support at the socket layer;
2. Mapping of client authentication to permissions on the server side;
3. PSK specification on the client side.

Most of the effort so far has been on the first point, with the addition and testing of support for GnuTLS to the OComm socket library. This is still a work in progress.
The server configuration code has also been overhauled in preparation for the client-mapping support. This led to the introduction and documentation of an XML configuration file covering all server option which could, up to now, only be specified on the command line. Support for PSK mapping is yet to be implemented.
On the client side, the PSK can be specified as part of the collection server URI, to be passed to the OComm socket upon instantiation.
The latest OML source code is available at: http://git.mytestbed.net/?p=oml.git

3.4 User-friendly data access and visualization

To describe the possibilities for users to access data in a user-friendly way from OML collection resources. Here two ways will be described: 1) SPARQL query based to retrieve data from semantic OML, and 2) Manifold query based to retrieve data from classic OML with PostgreSQL.

3.4.1 SPARQL

For the data to be accessed, in case the semantic OML is used and the Jena TDB as a backend triple store, the user can query the data through SPARQL query tool[3].
To enable data visualization, Sgvizler[4] tool is used in the implementation. The following steps are required to integrate Sgvizler tool with the Fuseki server:

- Download the source code from here http://dev.data2000.no/sgvizler/, but we don't need all the files, only the JavaScript file called 'sgvizler.js' is required.
- Put the JavaScript file in the Fuseki directory under 'pages'.
- You can then see it if you browse the Fuseki server pages.

A user can then query the data either via a terminal (through a command-line tool) or via a GUI.
To query all data from Fuseki via terminal, use the “s-query” which is located in the Fuseki directory:

```
/path/to/s-query  --service  http://<serverIP>:<serverPort>/<dataset>/query
'SELECT * {GRAPH <http://<serverIP>:<serverPort>/<domain>> {?s ?p ?o}}'
```

To open the GUI, open <fuseki-host>:<fuseki-port> in browser (e.g. localhost:8080), and under the Control Panel tap, choose the appropriate dataset and enter the query. But, if Sgvizler is installed, there should be a new link (e.g. SPARQL Visualization), where you can enter the query and choose what

[3] http://www.w3.org/TR/rdf-sparql-query/

[4] https://www.w3.org/2001/sw/wiki/Sgvizler

kind of charts you want to see. A query example to show a line chart of used bandwidth of a specific resource (cf. Figure 4):

```
SELECT ?timevalue ?value ?prefix ?unit{{
?measure omn-monitoring:isMeasurementDataOf ?metric .
?metric rdf:type omn-monitoring-metric:UsedMemory .
?metric omn-monitoring:isMeasurementMetricOf ?resource .
?measure omn-monitoring-data:hasMeasurementDataValue ?value .
?measure omn-monitoring:hasUnit ?unit_value .
?unit_value rdf:type ?unit .
?unit_value omn-monitoring-unit:hasPrefix ?prefix_value .
?prefix_value rdf:type ?prefix .
?measure omn-monitoring-data:hasTimestamp ?timevalue .
?resource omn:hasURI ?uri .
filter(regex(?uri,<resource-uri>))
}}
```

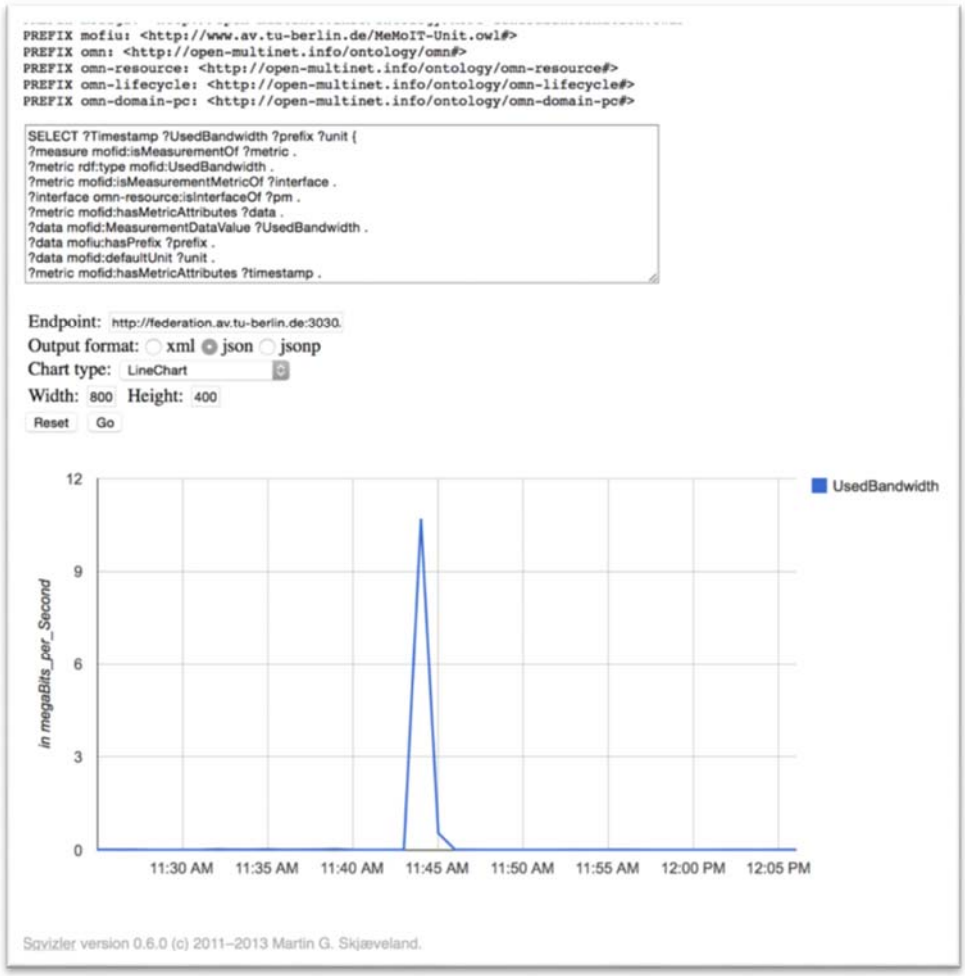

Figure 4: Example SPARQL Visualization of monitoring data

3.4.2 Manifold

As shown in Figure 5 the integration of Manifold with OML provides a single access to distributed OML databases relying on PostgreSQL. It provides three interfaces for different usages: **shell, Python**

and REST. The **shell** allows to quickly test a Query to check the values stored in the databases. The **python interface** uses an XML-RPC protocol. Using a remote python client allows to integrate Manifold Queries into another application. The last interface is a **REST API** that provides an easy access to data for web services for instance Reputation service and SLA management in the Fed4FIRE context.

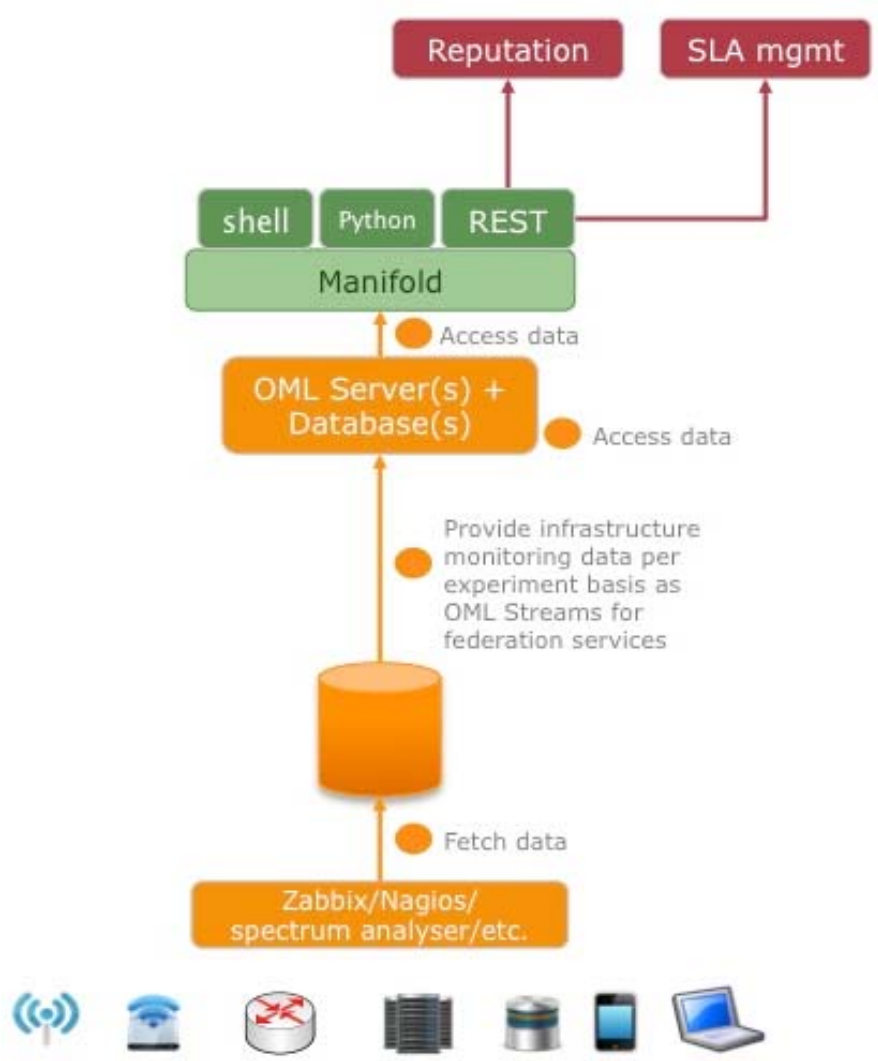

Figure 5: Manifold interfaces to retrieve monitoring data

3.4.2.1 Manifold Shell

The Manifold library has to be installed in order to use the Shell interface.

```
git clone git://git.onelab.eu/manifold.git
cd manifold
git checkout routerv2
make && make install
```

Then connect the Shell to the Manifold instance hosted for Fed4FIRE and the user can send Queries.

```
manifold-shell -x --url https://cetus.ipv6.lip6.fr:7080 -u xxx -p xxx
Welcome to MANIFOLD shell. Press ^C to clean up command line, ^D to exit.
manifold>>>
```

Sample of a Query to retrieve all the objects advertised by Manifold connected to OML and its result:

```
manifold>>> select object_name from local:object
===== RESULTS =====
[{'object_name': u'diskiops'},
 {'object_name': u'phys_load_5min'},
 {'object_name': u'phys_total_vms'},
 {'object_name': u'monitoring_for_federation'},
 {'object_name': u'_senders'},
 {'object_name': u'mem'},
 {'object_name': u'phys_load_1min'},
 {'object_name': u'storage'},
 {'object_name': u'diskiops_2'},
 {'object_name': u'cpu'},
 {'object_name': u'availability_2'},
 {'object_name': u'_experiment_metadata'},
 {'object_name': u'mem_2'},
 {'object_name': u'phys_availability'},
 {'object_name': u'memory'},
 {'object_name': u'cpu_2'},
 {'object_name': u'_availability'},
 {'object_name': u'runningvms_2'},
 {'object_name': u'runningvms'},
 {'object_name': u'availability'},
 {'object_name': u'phys_load_15min'},
 {'object_name': u'about'}]
```

Using Manifold Shell, the Query is very similar to SQL syntax:

```
SELECT field1, field2 FROM object WHERE field1==“value” && field2==“value”
UPDATE object SET field1=”value” WHERE field2==”value”
INSERT INTO object SET field1=”value”, field2=”value”
DELETE FROM object WHERE field1==“value”
```

3.4.2.2 Manifold Python

3.4.2.2.1 XML-RPC Client

Using the standard XML-RPC library of Python, a user can connect to the Manifold instance hosted for Fed4FIRE. It doesn't require any package installation and can be used in any Python application.

```
#!/usr/bin/env python
import xmlrpclib
srv = xmlrpclib.ServerProxy("https://cetus.ipv6.lip6.fr:7080/",
allow_none=True)
auth = {"AuthMethod": "password", "Username": "xxx", "AuthString": "xxx"}

q = {
   'action' : 'get',
   'object' : 'local:object',
   #'filters': [],
   'fields' : ["table"]
}
rs=srv.forward(q,{'authentication':auth})

print rs
```

The returned value would be the same as the example in 0

3.4.2.2.2 Action

Action value can be one of the followings:

`get | create | update | delete`

3.4.2.2.3 Object

Object value can be any of the object_name advertised by Manifold, as a FROM in SQL.
The object names can be retrieved with the sample Query provided in 0 and 3.4.2.2.1.

3.4.2.2.4 Filters

Filters is a list of filters to be applied as a WHERE clause in SQL.
A filter is composed of a field name, an operator and a value.

`[field, operator, value]`

A field name can be any of the properties of the object considered in the query.
Operator can be one of the followings:

`== | != | < | <= | > | >= | INCLUDED`

Value can be any text.

3.4.2.2.5 Fields

Fields is a list of field names among the properties of the object considered in the query.
The selection of fields allows restricting the result to a subset of properties of an object, as would do a SELECT statement in SQL.

3.4.2.3 Manifold REST

A REST API has been developed to provide an easy access to Manifold for any web service application through a simple URL.

3.4.2.3.1 Metadata

The local namespace allows retrieving metadata about the tables stored in the distributed databases. Databases tables are considered as objects in Manifold.

List of object names:

```
http://cetus.ipv6.lip6.fr/local:object?fields=object_name
```

List of field names for each object:

```
http://cetus.ipv6.lip6.fr/local:object?fields=object_name,name
```

3.4.2.3.2 Fields

The selection of fields allows restricting the result to a subset of properties of an object, which corresponds to columns of a table in a database.
We consider the availability of resources as an example. The availability object is composed of the following field names:

```
{"object_name": "availability",
      "name": [
              "node",
              "oml_sender_id",
              "sliverID",
              "oml_tuple_id",
              "experimentID",
              "up",
              "oml_seq",
              "oml_ts_client",
              "last_check",
              "oml_ts_server"
              ]
}
```

3.4.2.3.3 Filters

Applying filters to a Query allows to restrict the result to data matching some values.

As an example, the following Query using filters can be executed in order to get the availability data of the node007 from the Nitos testbed after 24th January 2016.

```
http://cetus.ipv6.lip6.fr/availability?filter=(node_in_['urn:publicid:IDN%2
Bomf:nitos.outdoor%2Bnode%2Bnode007'])and(last_check_gte_'2016-01-24')
```

If the user wants to restrict the result to some fields, he or she can apply the fields selection.

```
http://cetus.ipv6.lip6.fr/availability?fields=up,last_check&filter=(node_in
_['urn:publicid:IDN%2Bomf:nitos.outdoor%2Bnode%2Bnode007'])and(last_check_g
te_'2016-01-24')
```

4 Conclusion

This deliverable reports on the third Fed4FIRE development cycle concerning the measurement and monitoring services. Following the specifications reported in D6.4 [1] for the third cycle implementation, three main features are implemented to support secure data transportation and collection, following a common monitoring information model, and to allow users to access their data in a user-friendly manner. In more detail, a common information model was presented based on semantics which was enabled through the appropriate OML extensions in the client and server side. Furthermore, work is underway to enhance OML with security support regarding secure data transportation and collection. Finally, user-friendly data access and visualization was described based on the technologies of SPARQL queries and Manifold.

References

[1] Fed4FIRE D6.4 Detailed specifications regarding monitoring and measurement for third cycle ready. Available online at: http://www.fed4fire.eu/deliverables/.

[2] Fed4FIRE First Level Support Dashboard. Available online at: https://flsmonitor.fed4fire.eu.

[3] O. Mehani, G. Jourjon, T. Rakotoarivelo, and M. Ott, "An instrumentation framework for the critical task of measurement collection in the future Internet," Computer Networks, vol. 63, pp. 68-83, Apr. 2014.

[4] Fed4FIRE D2.7 Third federation architecture. Available online at http://www.fed4fire.eu/deliverables/.

[5] Fed4FIRE D8.7 Third input to WP2 concerning first level support. Available online at http://www.fed4fire.eu/deliverables/.

[6] Fed4FIRE D3.4 / D4.4 Third input from community to architecture. Available online at http://www.fed4fire.eu/deliverables/.

[7] A. Willner, C. Papagianni, M. Giatili, P. Grosso, M. Morsey, Y. Al-Hazmi, and I. Baldin, “Open-Multinet Upper Ontology — Towards the Semantic-based Management of Federated Infrastructures,” in 10th International Conference on Testbeds and Research Infrastructures for the Development of Networks & Communities (TRIDENTCOM 2015), Vancouver, Canada, ACM, 2015, pp. 1–10. doi: 10.4108/icst.tridentcom.2015.259750.

[8] G. Klyne, J. J. Carroll, and B. McBride, “Resource description framework (RDF): Concepts and abstract syntax,” W3C, W3C Recommendation, 2004.

[9] D. Brickley and R. V. Guha, “Resource Description Framework (RDF) Schema 1.1. W3C Recommendation,” World Wide Web Consortium, 2014.

[10] Semantic OML. Online at: https://github.com/alhazmi/semantic-oml.

[11] OML. Online at http://oml.mytestbed.net/projects/oml/wiki/BuildingSource.

[12] Jena Apache Fuseki server. Online at http://jena.apache.org/documentation/serving_data/#getting-started-with-fuseki.