Grant Agreement No.: 732638
Call: H2020-ICT-2016-2017
Topic: ICT-13-2016
Type of action: RIA

D3.2: Developments for the first cycle

Work package	WP 3
Task	Task 3.1-3.5
Due date	30/06/2018
Submission date	12/11/2018
Deliverable lead	Imec
Version	4
Authors	Brecht Vermeulen (imec), Wim Van der Meerssche (imec), Thijs Walcarius (imec), Radomir Klacza (SU), Pauline Gaudet Chardonnet (SU), Dimitris Dechouniotis (NTUA), Costas Papadakis (NTUA), Aris Dadoukis (CERTH), Donatos Stavropoulos (CERTH), Ana Juan Ferrer (ATOS), Roman Sosa Gonzalez (ATOS), Joaquin Iranzo Yuste (ATOS), Rowshan Jahan Sathi (TUB), Alex Willner (TUB), Lucas Nussbaum (Inria), David Margery (Inria), Cedric Crettaz (MI)
Reviewers	Peter Van Daele (imec)

Abstract	This deliverable gives an overview of the developments in WP3 during the first 18 months of the project. WP2 are normal operations developments (add testbeds, fix bugs, small features, etc). WP3 is focussing on larger new functionality.
Keywords	Developments first cycle, new functionality

Document Revision History

Version	Date	Description of change	List of contributor(s)
V1	1/06/2018	TOC	Brecht Vermeulen (imec)
V2	24/10/2018	First complete version	Brecht Vermeulen (imec), Wim Van der Meerssche (imec), Thijs Walcarius (imec), Radomir Klacza (SU), Pauline Gaudet Chardonnet (SU), Dimitris Dechouniotis (NTUA), Costas Papadakis (NTUA), Aris Dadoukis (CERTH), Donatos Stavropoulos (CERTH), Ana Juan Ferrer (ATOS), Roman Sosa Gonzalez (ATOS), Joaquin Iranzo Yuste (ATOS), Rowshan Jahan Sathi (TUB), Alex Willner (TUB), Lucas Nussbaum (Inria), David Margery (Inria), Cedric Crettaz (MI)
V3	8/11/2018	Almost final version	Brecht Vermeulen (imec), Wim Van der Meerssche (imec), Thijs Walcarius (imec), Radomir Klacza (SU), Pauline Gaudet Chardonnet (SU), Dimitris Dechouniotis (NTUA), Costas Papadakis (NTUA), Aris Dadoukis (CERTH), Donatos Stavropoulos (CERTH), Ana Juan Ferrer (ATOS), Roman Sosa Gonzalez (ATOS), Joaquin Iranzo Yuste (ATOS), Rowshan Jahan Sathi (TUB), Alex Willner (TUB), Lucas Nussbaum (Inria), David Margery (Inria), Cedric Crettaz (MI)
V4	12/11/2018	Final version	Brecht Vermeulen (imec), Wim Van der Meerssche (imec), Thijs Walcarius (imec), Radomir Klacza (SU), Pauline Gaudet Chardonnet (SU), Dimitris Dechouniotis (NTUA), Costas Papadakis (NTUA), Aris Dadoukis (CERTH), Donatos Stavropoulos (CERTH), Ana Juan Ferrer (ATOS), Roman Sosa Gonzalez (ATOS), Joaquin Iranzo Yuste (ATOS), Rowshan Jahan Sathi (TUB), Alex Willner (TUB), Lucas Nussbaum (Inria), David Margery (Inria), Cedric Crettaz (MI)

ACKNOWLEDGMENT

Co-funded by the European Union

Co-funded by the Swiss Confederation

This deliverable has been written in the context of a Horizon 2020 European research project, which is co-funded by the European Commission and the Swiss State Secretariat for Education, Research and Innovation. The opinions expressed and arguments employed do not engage the supporting parties.

Project co-funded by the European Commission in the H2020 Programme		
Nature of the deliverable:	**R**	
Dissemination Level		
PU	Public, fully open, e.g. web	✓
CL	Classified, information as referred to in Commission Decision 2001/844/EC	
CO	Confidential to FED4FIRE+ project and Commission Services	

** R: Document, report (excluding the periodic and final reports)*

DEM: Demonstrator, pilot, prototype, plan designs

DEC: Websites, patents filing, press & media actions, videos, etc.

OTHER: Software, technical diagram, etc.

EXECUTIVE SUMMARY

This deliverable gives an overview of the developments in WP3 during the first 18 months of the project. WP2 are normal operations developments (add testbeds, fix bugs, small features, etc). WP3 is focussing on larger new functionality.

WP3 consists out of the following tasks, which are also the sequence of sections in this deliverable:

- Task 3.1 is focussing on SLA and reputation for testbed usage
- Task 3.2 is focussing on Experiment-as-a-Service (EaaS), data retention and reproducibility of experiments
- Task 3.3 is targeting Federation monitoring and interconnectivity
- Task 3.4 works on Service orchestration and brokering
- Task 3.5 researches ontologies for the federation of testbeds

Co-funded by the Horizon 2020 Framework Programme of the European Union

TABLE OF CONTENTS

Co-funded by the Horizon 2020
Framework Programme of the European Union

Co-funded by the Horizon 2020
Framework Programme of the European Union

Co-funded by the Horizon 2020
Framework Programme of the European Union

LIST OF FIGURES

Co-funded by the Horizon 2020
Framework Programme of the European Union

Co-funded by the Horizon 2020 Framework Programme of the European Union

Co-funded by the Horizon 2020
Framework Programme of the European Union

Co-funded by the Horizon 2020 Framework Programme of the European Union

LIST OF TABLES

Co-funded by the Horizon 2020 Framework Programme of the European Union

ABBREVIATIONS

FIRE	Future Internet Research and Experimentation
JSON	JavaScript Object Notation
SLA	Service Level Agreement
SLO	Service Level Objective
XML	eXtensible Markup Language
WSAG	Web Service-Agreement
API	Application Programming Interface
XML-RPC:	Extensible Markup Language Remote procedure call
REST	REpresentational State Transfer
AM	Aggregate Manager
QoS	Quality of Service
QoE	Quality of Experience
MVC	Model-View-Controller
O/RM	Object-relational mapping
GUI	Graphical User Interface
CLI	Command Line Interface
HRS	Hybrid Reputation System
KPI	Key Performance Indicator
FAHP	Fuzzy Analytic Hierarchical Process

Co-funded by the Horizon 2020
Framework Programme of the European Union

1 INTRODUCTION

This deliverable gives an overview of the developments in WP3 during the first 18 months of the project. WP2 are normal operations developments (add testbeds, fix bugs, small features, etc). WP3 is focussing on larger new functionality.

WP3 consists out of the following tasks, which are also the sequence of sections in this deliverable:

- Task 3.1 is focussing on SLA and reputation for testbed usage
- Task 3.2 is focussing on Experiment-as-a-Service (EaaS), data retention and reproducibility of experiments
- Task 3.3 is targeting Federation monitoring and interconnectivity
- Task 3.4 works on Service orchestration and brokering
- Task 3.5 researches ontologies for the federation of testbeds

2 SLA AND REPUTATION SERVICE

The overall objective of SLA management in FED4FIRE+ is to provide the capability to testbed providers and experimenters to establish agreements with regards to the use of the experimentation infrastructure, validate the enactment of these agreements; as well as; to be notified in cases of not fulfilment of the agreements. The functional requirements of this component, its fit into the FED4FIRE+ overall architecture and initial details on its internal architecture were already presented in D3.01 Requirements and Specifications for the first cycle. In this deliverable we further elaborate on its internal design and we present in detail its implementation for the first cycle together with specification of APIs provided installation and user guides.

Since FED4FIRE+ provides many heterogeneous testbeds with similar resources, the selection of the appropriate resources by the experimenters becomes a tedious task. Thus, a trust mechanism between the testbed providers and FED4FIRE+ users must be established in order to facilitate the testbed and resources selection. The trust is defined as the subjective belief of entity A, that entity B performs a given action [1]. Reputation is a complementary concept that helps entities to trust each other. Reputation is defined as “the general belief about a person’s or thing’s character or standing”, according to Concise Oxford Dictionary. In this deliverable, we present the details of the new reputation algorithm and the progress on the development of the reputation service according to the functional requirements of D3.01 Requirements and Specifications for the first cycle.

Figure 1 illustrates the overall architecture of SLA and Reputation Service of FED4FIRE+ project. As shown below, the experimenter should interact with the Reputation Service through jFed or the Portal (MySlice v2) at the end of the conducted experiment by sending his/her evaluation for every testbed used in the experiment through the GUI. The Reputation Service will retrieve the data needed for the reputation score calculation from every testbed used through monitoring data APIs and through the SLA collector as explained more thoroughly in the sections below. Every testbed must provide a REST API for monitoring data and install the SLA components.

Co-funded by the Horizon 2020
Framework Programme of the European Union

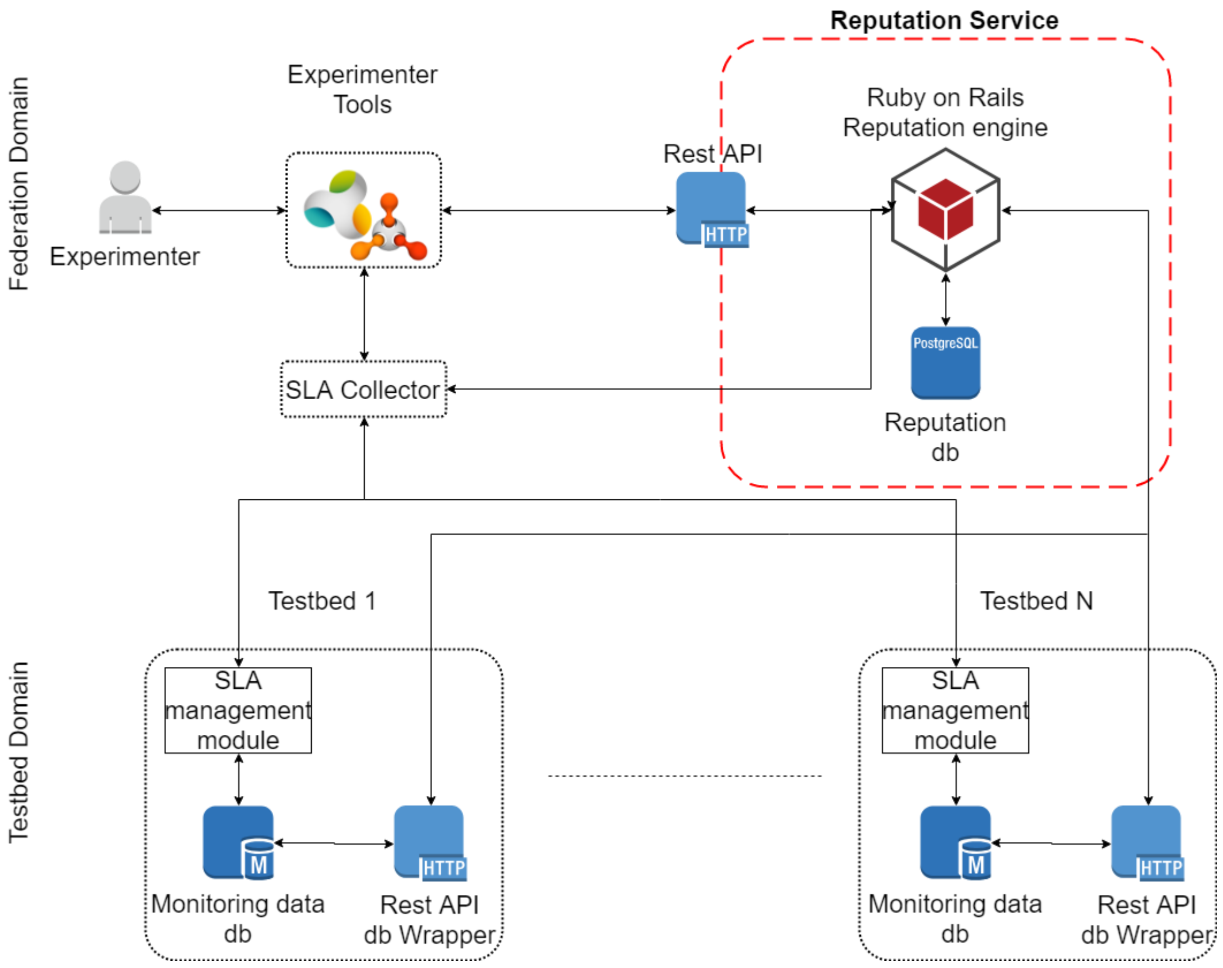


Figure *1:* FED4FIRE+ SLA and Reputation Service Architecture

2.1 SLA DEVELOPMENTS FOR THE FIRST CYCLE

2.1.1 Detailed Architecture

The SLA Component internal architecture is detailed in Figure 2. The architecture details the three main components of this component: SLA Dashboard, GUI that permits testbed providers to define agreements; SLA Collector, which interfaces with each testbed monitoring data to collect metrics and allows subscription of other components to receive violations and penalties resultant of not fulfillment of the agreed SLAs ; and the SLA management module (SLA Core) responsible of implementing the business logic of validating metrics from existing agreements and raising alarms in agreement breach situations.

Co-funded by the Horizon 2020 Framework Programme of the European Union

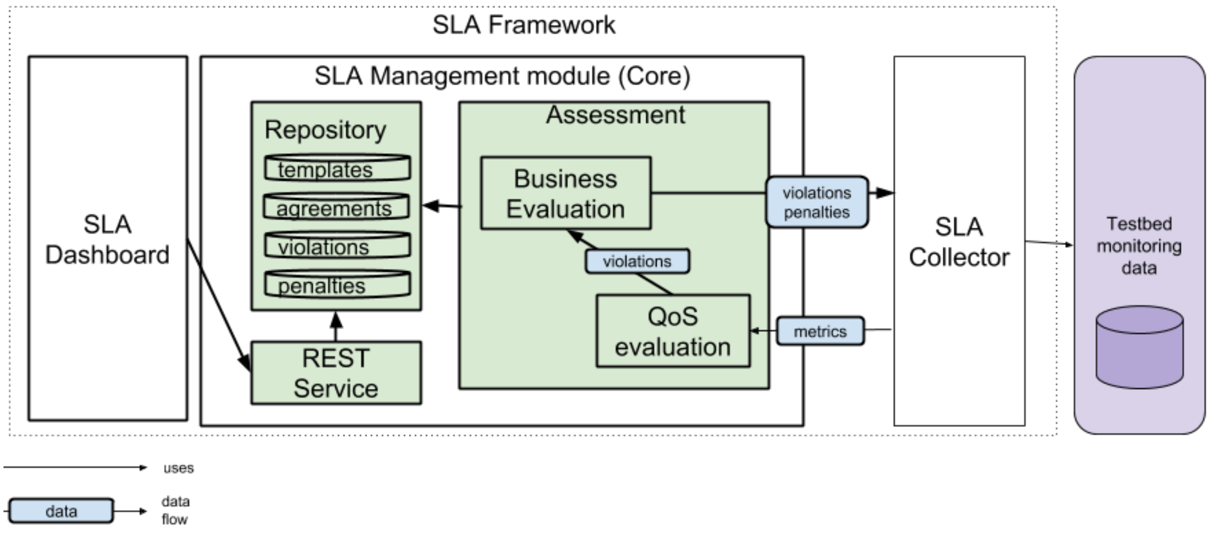


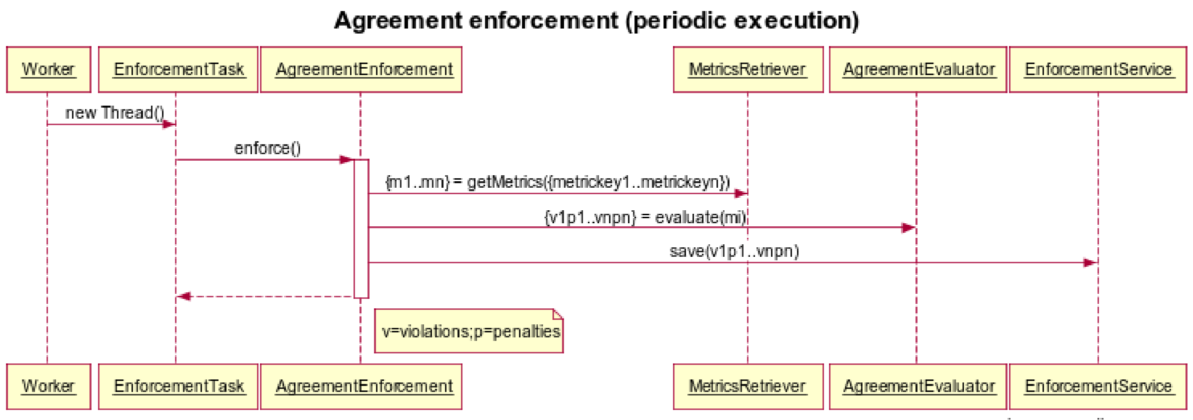
Figure 2: SLA Framework internal architecture

2.1.2 Sequence Diagrams

The following sequence diagrams detail the main operations performed in the SLA Management module (Core) in order to evaluate an SLA.

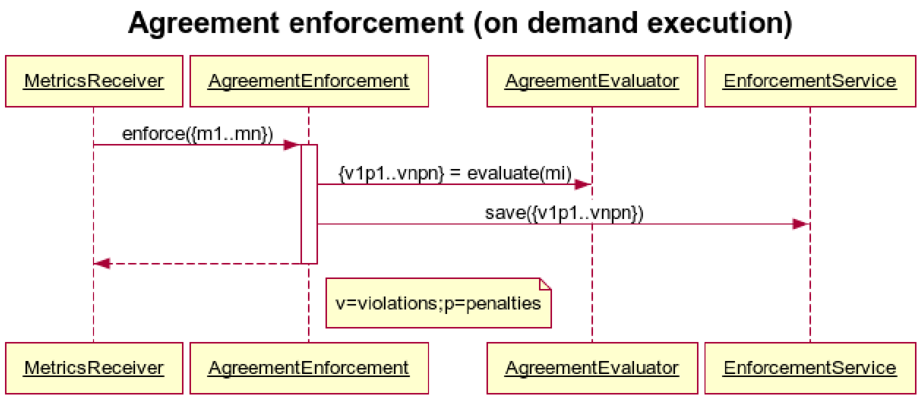
SLA enforcement process details the steps used to evaluate that an established agreement is fulfilled by a provided. More in detail, this is the process in which collected monitoring values are assessed to understand if they fulfil the provided constraints. Depending on configuration of the core, the evaluation process happens on demand or at a certain periodicity period.

2.1.2.1 Periodic agreement enforcement

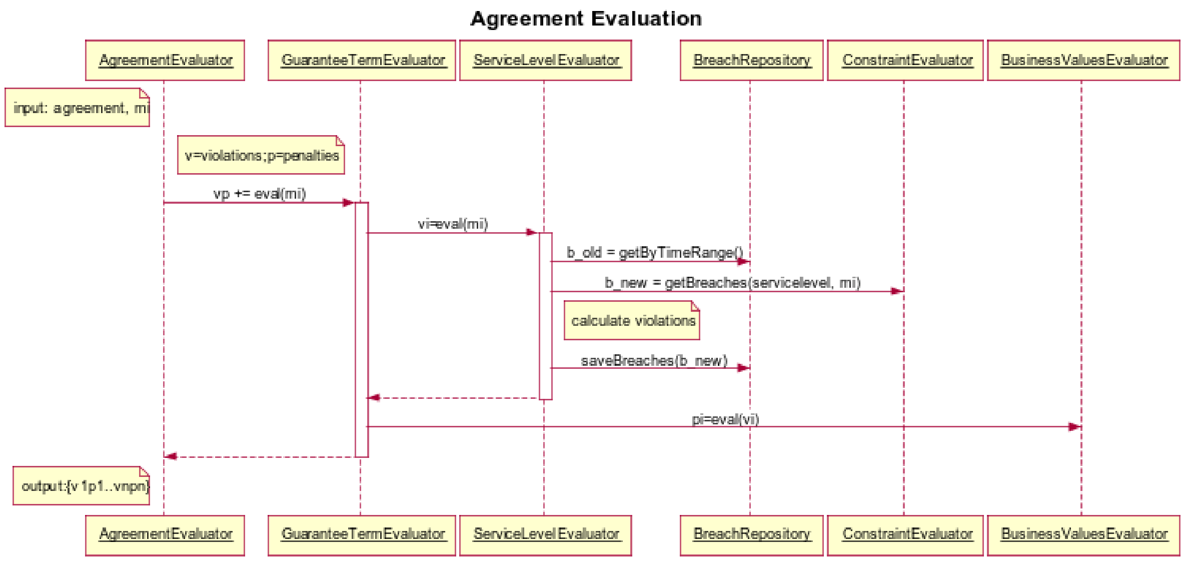


Co-funded by the Horizon 2020 Framework Programme of the European Union

2.1.2.2 On demand agreement enforcement



2.1.2.3 Agreement evaluation



The following classes are used in the SLA evaluation process:

AgreementEnforcement: collects by means of the SLACollector the configured metrics necessary to measure the enforcement of an agreement. It interfaces with AgreementEvaluator to detect if a violation or non-fulfilment case has happened. In this case, collected values are stored in its internal repository.

AgreementEvaluator: invokes GuaranteeTermEvaluator per all guarantee terms in the agreement.

Co-funded by the Horizon 2020 Framework Programme of the European Union

GuaranteeTermEvaluator: calls ServiceLevelEvaluator to obtain triggered violations; in this case ot invokes BusinessValuesEvaluator using the violations as input.

ServiceLevelEvaluator: computes generated violations. Diverse evaluators can be used. In Fed4FIRE+ the followed approach is based on policies. The policy determines the repetitions a violation metric has to occur in a certain time period so to incur into a violation.

ConstraintEvaluator. Per each agreed term, it isolates the service level constraint, and assesses if a collected monitoring metric adheres to the defined constraint.

MetricsRetriever. It performs periodic queries to collect last metrics obtained by an agreement. It is used in periodic execution evaluation.

MetricsReceiver. It invokes SLA Collector to receive last metrics gathered from an agreement. It is used in on-demand execution evaluation.

2.1.3 Requirements Coverage in Iteration 1

Table 1 presents the SLA Components requirements defined in D3.01 Requirements and Specifications deliverable, providing explanations on the maturity achieved in the first cycle and expected related next steps. These are later summarised in section 1.4 Future work. Requirements indicated in green are considered to be completed with provided version in first cycle.

Table *1:* SLA Functional Requirements

ID	Title	Coverage in First cycle
SLA_01	SLA solution must cover the whole lifecycle specified in WS-Agreement	The first cycle implementation fulfils this requirement.
SLA_02	SLA solution REST interface	The rest interface is provided and detailed in next subsection of this deliverable.
SLA_03	SLA solution Subscription mechanism	In this first cycle metrics collection mechanism has yet to support subscription.
SLA_04	SLA solution multitenant	The SLA solution supports multitenancy by being capable of managing diverse providers without interference. Additional development will be required in order to synchronize databases among testbeds.

Co-funded by the Horizon 2020 Framework Programme of the European Union

SLA_05	SLA Dashboard	A basic SLA Dashboard exist for first cycle. This has yet to be integrated with new version of MySlide.
SLA_06	Agreement creation and enactment	This requirement is contained into previous requirements SLA_01. Therefore, it is achieved in first cycle, representing basic component functionality
SLA_07	SLA terms quantizable	For this initial cycle testbeds making use of the SLA management (NTUA and Nitos) have decided to overall availability metric. Further refinements of this metrics will require of validation.
SLA_08	SLA access to monitoring data	SLA management is able to collect metrics from the two testbeds in which it is installed. Additional federation mentioning support requires of adapting existing implementation to python and will be performed in upcoming cycles .
SLA_09	Distributed federation architecture.	As detailed in previous sections, the SLA is integrated into the Fed4FIRE+ Federated architecture, having initial deployments in two of the existing testbeds.
SLA_10	SLA solution software dependencies	SLA Management solution provides REST interfaces which makes them available regardless their programming language being python or java.

2.1.4 Developments and documentation

Table 2 summarizes the software requirements of the SLA Developments in the first iteration.

Table *2:* SLA Software Requirements

Components	Technologies

Co-funded by the Horizon 2020 Framework Programme of the European Union

SLA Dashboard – SLA collector	Mysql >= 5.0 Python >= 2.7
SLA Manager	Mysql >= 5.0 Oracle JDK >=1.7

Detailed documentation has been produced for this first cycle development which is provided as Annex 1 to this document. This documentation includes:

- SLA which provides detailed instructions about APIS provided and their usage.
- SLA User Guide, detailing steps for use and configuration of provided software
- SLA Installation Guide, addressed to testbed owners so to facilitate component installation guidance.

2.2 REPUTATION ALGORITHM

The Reputation Service of the Fed4Fire project was based on FTUE reputation framework [2]. For a conducted experiment, this framework used some predefined QoS metrics, e.g., Node Availability, and two QoE metrics, named Overall Experience and Quality, to update the reputation score per testbed per service. Furthermore, the credibility of the users was measured and considered on the final computation of the reputation score. The computation of the experimenter's credibility was based on the difference between the measured QoS metrics and the opinion of the user. The QoS and QoE variables had a specific set of numeric values, i.e., 1-5. Furthermore, no SLA data were used on the computation of the reputation score.

In the context of FED4FIRE+ and based on the FTUE framework, we implement a new reputation algorithm with better malicious user filtering. The new Hybrid Reputation Service (HRS) is actually a multi-criteria decision-making system with hierarchical structure, which can easily scale up. HRS leverages several QoS key performance indicators (KPIs), such as Node Availability, Link Availability and Server Availability, and QoE KPIs, e.g. Usability, Document Readability, testbed owner's support, in a hierarchical and scalable structure. The QoE KPIs are expressed by fuzzy variables. Fuzzy logic is suitable to express the nature of QoE KPIs. Thus, a set of linguistic values, such as 'Poor', 'Good' and 'Excellent', will be used for the user input and these terms will be mapped to the corresponding fuzzy values. Furthermore, FED4FIRE+ reputation service exploits SLA data to accurately update the experimenter's credibility score. This new service is able to provide an overall reputation score per testbed or an individual score per service per testbed. HRS is implemented in the Reputation Engine of SLA and Reputation Service in Figure 1.

HRS is based on the principles of Fuzzy Analytic Hierarchical Process (FAHP) [3]. FAHP is widely used on various cases, such as product design, operational research and cloud services [4]. FAHP is a ranking method based on numeric QoS and fuzzy QoE KPIs, e.g., node

Co-funded by the Horizon 2020 Framework Programme of the European Union

availability and support satisfaction respectively. In order to compute the reputation score of federated testbeds, several modifications are required. There are three key differences with respect to FAHP between our HRS, and the provider selection use cases, such as in [5]. First, our approach allows the users to assign their weights on the criteria according to their experiment's goals. Secondly, we compare the evaluation of an experiment conducted on a testbed with an ideal rating of a virtual user, which contains the best values of all criteria. Finally, the experimenter's credibility is considered in the computation of the reputation score in order to ensure the fair judgment of testbeds. In the following, the phases of the proposed HRS are analytically described.

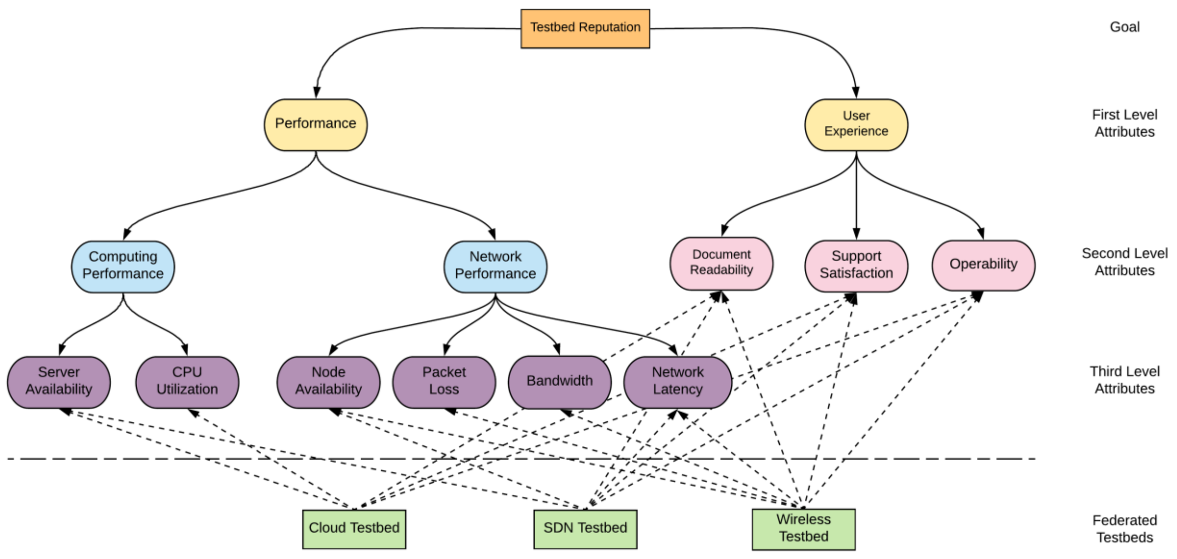


Figure *3:* HRS Model for FED4FIRE+ Testbeds

Phase 1 - Selection of testbed KPIs

The testbed owner determines the technical (QoS) and the user experience (QoE) KPIs and attributes that are used in the computation of the reputation score of the testbeds. Figure 3 shows a possible hierarchical structure with KPIs and attributes and highlights which of them are provided by each testbed. The difference between KPIs and attributes is that a KPI measures a specific technical or experience metric, while an attribute summarizes several KPIs of relevant metrics. At a specific level, the attributes can be further decomposed into the sibling attributes or KPIs of the lower level, while the KPIs cannot be decomposed further. Adopting SMICloud approach [6], numerical KPIs and attributes are represented by numeric, Boolean, unordered sets and range values. On the contrary, the QoE KPIs are represented by fuzzy numbers. In Figure 3, pink (right) and purple (left) colored KPIs refer to fuzzy and numerical attributes respectively. Table 1 contains the linguistic terms and the membership functions of the fuzzy numbers used for QoE attributes.

Co-funded by the Horizon 2020 Framework Programme of the European Union

Phase 2 - Weight Assignment

FED4FIRE+ experimenters use the federated testbed possibly focusing on different objectives. Thus, the importance of each attribute in the hierarchical structure is assigned by the experimenters themselves in a flexible manner. In the hierarchical model of Figure 3, each edge between two nodes has a weight that reflects the importance of the lower level attribute or KPI on the computation of the upper level attribute's value. The value of a weight is positive and less than one, while the sum of the assigned weights of the edges that links a group of nodes with their ancestor node is equal to one. Since the weights are derived from the subjective preferences of individuals, the final computation of reputation can still be based on inconsistent and conflicting KPIs and attributes. Thus, in order to avoid such inconsistencies,

Table *3:* Linguistic terms and Membership Functions of Fuzzy numbers

Linguistic Term	Membership Function
Very Poor (VP)	(1, 2, 3)
Poor (P)	(3, 4, 5)
Medium (M)	(4, 5, 6)
Good (G)	(5, 6, 7)
Very Good (VG)	(6, 7, 8)
Excellent (E)	(7, 8, 9)

the Consistency Ratio (CR) [7] is calculated for each group of sibling attributes. The CR is the degree of the randomness in the weight assignment between several sibling attributes. CR values less than 0.1 are acceptable to continue to the next phase, otherwise the experimenter must correct the assigned weights.

Phase 3 - Computation of relative attribute importance

After the conduction of an experiment, the user submits his rating for the QoS and QoE KPIs for all the participating testbeds. These ratings of QoS KPIs are modified taking into account the credibility of the experimenter, as it is analyzed in the following subsection III. The modified experimenter's ratings are compared against the ideal rating of a virtual user. The ideal rating is used to measure the distance between the actual performance of a testbed during an experiment and its perfect performance according to the experimenter's preferences. This is achieved by computing the Relative Attribute Comparison Matrix (RACM) for each KPI and attribute of the hierarchical model. Given the ideal rating A_V and the modified user's rating $\tilde{A}_u$ for the *X* KPI or attribute, $RACM_X$ is defined as follows,

$$RACM_X = \begin{bmatrix} 1 & \tilde{A}_u/A_u \\ A_u/\tilde{A}_u & 1 \end{bmatrix}$$

Co-funded by the Horizon 2020 Framework Programme of the European Union

If any KPI or attribute is numerical, the division of the ratings in RACM$_A$ is done according to Table 1 of [5, Section 3]. Otherwise, for fuzzy KPIs and attributes, the arithmetic operations on fuzzy numbers of [5, Section 3.1] are used.

Phase 4 - Computation and update of reputation

In the case of numerical KPIs and attributes, the extended AHP approach, similarly to SMICloud, is applied. For the fuzzy KPIs, the extended analysis on FAHP is adopted according to Chang's approach [3]. The combination of these methodologies uses the RACM of each KPI and attribute at any of the hierarchical model in order to calculate the score vector of all intermediate attributes and the top level reputation attribute. For the fuzzy RACMs, the following steps of extent analysis on FAHP [13] are applied. Let the N-dimension fuzzy $RACMA_A = [a_{ij}], i, j = 1, \dots, N$, , the fuzzy synthetic extent is defined by,

$$D_i = (D_i^l, D_i^m, D_i^u) = \sum_{j=1}^{N} a_{ij} \otimes \left(\sum_{i=1}^{N} \sum_{j=1}^{N} a_{ij} \right)^{-1}$$

We find the attribute with the higher fuzzy synthetic degree by computing the degree of possibility for a fuzzy number to be greater than other one,

$$V(D_i \geq D_j) = hgt(D_i \cap D_j) = \begin{cases} 1 & if D_i^m \geq D_j^m \\ \dfrac{D_j^l - D_i^u}{(D_i^m - D_i^u) - (D_j^m - D_j^l)} & if D_i^m \leq D_j^m \wedge D_j^l \leq D_i^u \\ 0 & otherwise \end{cases}$$

The degree of possibility that a fuzzy synthetic extent D$_i$ is greater than the rest synthetic fuzzy extents of the fuzzy RACM is,

$$d_i = V(D_i \geq D_k, \forall\, k = 1, \dots, N, k \neq i) = minV(D_i \geq D_j)$$

Finally, the normalized comparison vector is obtained,

$$c = [c_1 \dots c_N]^T where c_i = \frac{d_i}{\sum_{k=1}^{N} d_k}$$

At any level of the testbed's hierarchical model, we calculate the comparison vector for each attribute with the following bottom-up procedure. Given the weights of Phase 2, the ratings of the experimenter and the ideal rating, we start from the level, where KPIs exist, compute the comparison vector of the parent attribute by the comparison vectors of the sibling KPIs or attributes. Assuming a parent attribute with M sub-attributes and the weight vector with M elements, the comparison vector of the parent attribute is defined,

$$c_{par} = \begin{bmatrix} c_1^{tt} & \cdots & c_{subM}^{tt} \\ c_1^u & \dots & c_{subM}^u \end{bmatrix} \begin{bmatrix} w_1 \\ \vdots \\ w_{subM} \end{bmatrix} = \begin{bmatrix} c_{par}^{tt} \\ c_{par}^u \end{bmatrix}$$

Co-funded by the Horizon 2020
Framework Programme of the European Union

Reaching the top level of the hierarchical model, the normalized comparison vector for the Testbed Reputation attribute is computed, $c_{rep} = [c_{rep}^{\tilde{u}} \quad c_{rep}^{v}]^T$. The first element of this vector refers to the experimenter evaluation, while the second corresponds to the best possible rating of the virtual user. The difference between the two elements indicates the distance between the actual performance as interpreted by the experimenter, and the perfect performance of the testbed. Thus, for the nth conducted experiment, the testbed's reputation score is computed by,

$$R_{exp}^{T} = \frac{c_{rep}^{\tilde{u}}}{c_{rep}^{u}} 100\%$$

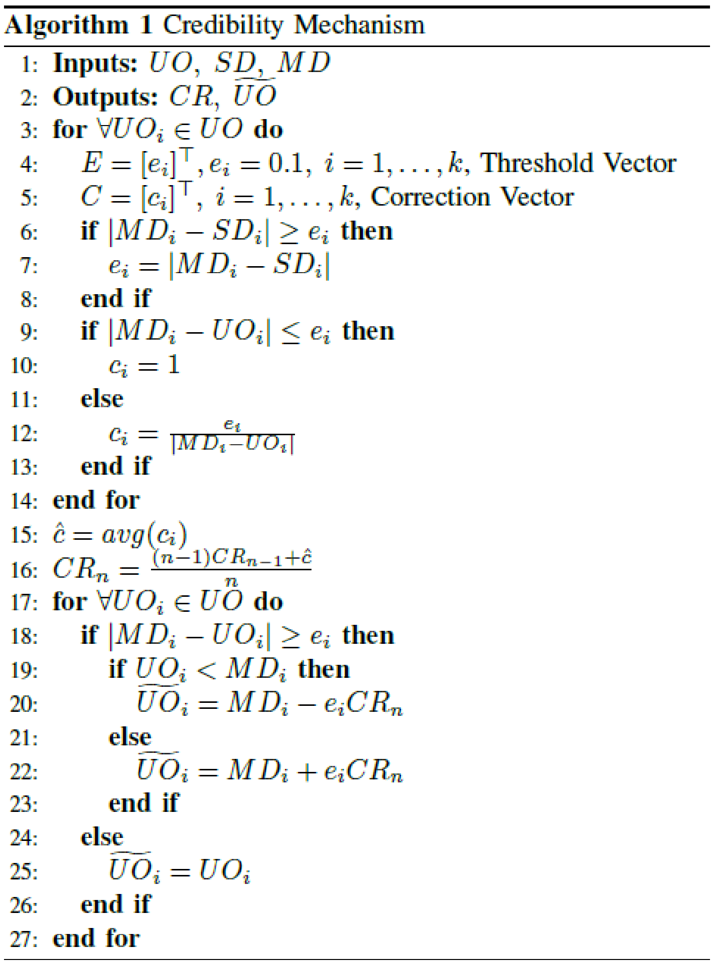
After *n* completed experiments, the overall reputation value of the testbed is updated,

$$R_{n}^{T} = \frac{(n-1)R_{n-1}^{T} + R_{exp}^{T}}{n}$$

Credibility Mechanism

The credibility mechanism, developed for HRS, aims at reducing the impact of malicious users in the computation of reputation score. The credibility mechanism takes into account the QoS KPIs and the respective SLA value. Essentially, the experimenter's subjective opinion, the predefined SLA and the monitoring value are compared in order to check the divergence between the rating and the testbed's actual performance. In this process the non-technical QoE KPIs are excluded due to their subjective nature.

Figure 4 shows the credibility mechanism in algorithmic fashion. The represented process concerns the use of one testbed for one experiment. Nevertheless, in an experiment procedure, users can use more than one testbeds. In that case the experimenter's credibility value is sequentially calculated for every testbed. Considering the user's opinion (ratings) for every QoS KPI, as the vector $UO = [UO_i]^T, i = 1, \dots, k$ and the corresponding vectors for the monitoring data, $MD = [MD_i]^T, i = 1, \dots, k$ and SLA data, $SD = [SD_i]^T, i = 1, \dots, k$, the algorithm computes the updated credibility value for the specific experimenter and a vector with the updated ratings for the QoS KPIs as those modified by the credibility mechanism, $\tilde{U}O = [\tilde{U}O_i]^T, i = 1, \dots, k$ respectively. For each KPI of the testbed, the threshold and correction vectors are initialized (lines 4-5). The elements of the threshold vector express the tolerance against an opinion and is based on the deviation of the monitoring data from the SLA reference value (lines 6-13). The elements of the correction vector are actually a credibility value for each KPI. The user's credibility for an experiment is calculated as the average value of the correction vector. Then, the overall user's credibility is updated according to lines 15-16. For each KPI, we adapt the experimenter's opinion if the difference between the opinion and the monitoring data is greater than the respective threshold value. The modified opinion is based on the monitoring value, the updated user's credibility and the threshold value (lines 17-27). The modified opinions on KPIs are used in Phase 3 of HRS.



```
Algorithm 1 Credibility Mechanism
1:  Inputs: $UO,\ SD,\ MD$
2:  Outputs: $CR,\ \widetilde{UO}$
3:  for $\forall UO_i \in UO$ do
4:      $E = [e_i]^\top, e_i = 0.1,\ i = 1, \dots, k$, Threshold Vector
5:      $C = [c_i]^\top,\ i = 1, \dots, k$, Correction Vector
6:      if $|MD_i - SD_i| \geq e_i$ then
7:          $e_i = |MD_i - SD_i|$
8:      end if
9:      if $|MD_i - UO_i| \leq e_i$ then
10:         $c_i = 1$
11:     else
12:         $c_i = \frac{e_i}{|MD_i - UO_i|}$
13:     end if
14: end for
15: $\hat{c} = avg(c_i)$
16: $CR_n = \frac{(n-1)CR_{n-1} + \hat{c}}{n}$
17: for $\forall UO_i \in UO$ do
18:     if $|MD_i - UO_i| \geq e_i$ then
19:         if $UO_i < MD_i$ then
20:             $\widetilde{UO}_i = MD_i - e_i CR_n$
21:         else
22:             $\widetilde{UO}_i = MD_i + e_i CR_n$
23:         end if
24:     else
25:         $\widetilde{UO}_i = UO_i$
26:     end if
27: end for
```

Figure *4:* Credibility Mechanism of HRS

2.3 REPUTATION DEVELOPMENTS FOR THE FIRST CYCLE

As described in the previous section, HRS has been developed and deployed as a prototype version for early integration and testing for the first cycle. The Reputation Service has been deployed as a centralized service on the Ruby on Rails MVC Framework. The architecture of this testing phase is depicted in Figure 5. For the early testing phase, the NETMODE and NITOS testbed were used. At this stage, the Reputation Service is not interconnected with the platform as a whole. The testing and execution of the experiment evaluation lifecycle was conducted through REST API calls with no Reputation Service frontend tools.

Co-funded by the Horizon 2020 Framework Programme of the European Union

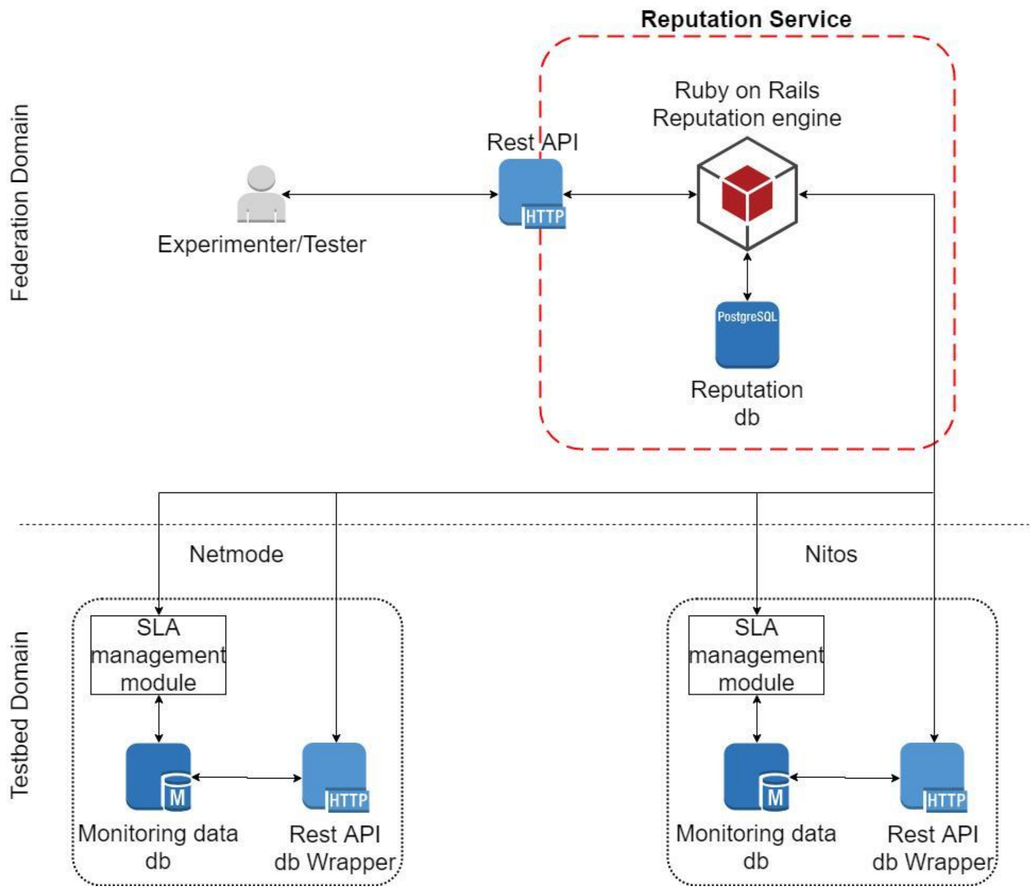
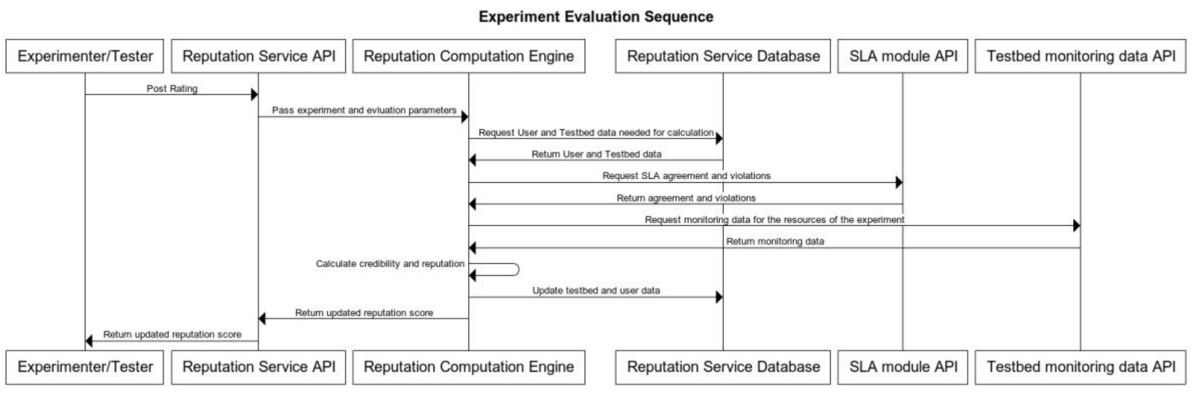


Figure 5: Reputation Service Architecture

The rating lifecycle goes as follows:

After the completion of an experiment, the experimenter evaluates the testbeds used in the experiment and the ratings are submitted to the Reputation Service through the Rest API. Then, the Reputation Service Engine retrieves the SLA agreement and the monitoring data for the utilized resources and updates the involved testbeds' reputation score and the user's credibility as described previously. Finally, the results are stored in the Reputation Service Database and the updated reputation value is returned to the experimenter as a response to the initial API call. The above procedure is shown also in the sequence diagram below.

Co-funded by the Horizon 2020 Framework Programme of the European Union


Experiment Evaluation Sequence
Experimenter/Tester
Reputation Service API
Reputation Computation Engine
Reputation Service Database
SLA module API
Testbed monitoring data API
Post Rating
Pass experiment and evluation parameters
Request User and Testbed data needed for calculation
Return User and Testbed data
Request SLA agreement and violations
Return agreement and violations
Request monitoring data for the resources of the experiment
Return monitoring data
Calculate credibility and reputation
Update testbed and user data
Return updated reputation score
Return updated reputation score

As shown by the testing architecture layout, the only requirement to integrate a testbed with the reputation service is to install the SLA components following the instructions provided in Appendix A. Furthermore, a Rest API must be developed or installed as a wrapper for the monitoring data database. Both steps have been completed and tested in NITOS and NETMODE testbeds. Although the SLA installation is no testbed specific, minor modifications are required for each testbed in order to evaluate the monitoring metrics, e.g., availability, and configure it to retrieve the appropriate monitoring data from the monitoring data API. For the monitoring data APIs the solution differs for each testbed. The different approaches depend on the database each testbed uses for storing monitoring data. In this cycle two wrappers have been used. One developed by NITOS for MySql database and one in NETMODE for PostgreSQL database with PostgREST API installed. Tables 4-6 show the utilized software tools for the Reputation Service and the monitoring REST APIs in NETMODE and NITOS testbeds.

Table *4:* Reputation service requirements

Components	Technologies
Reputation Service API and Computation Engine	Rails = 5.1.4 Ruby >= 2.2.2
Reputation Service Database	PostgreSQL = 10.3

Table 5: NITOS monitoring Rest API

Components	Technologies
Monitoring Data Database	MySql 5.5.32

Co-funded by the Horizon 2020 Framework Programme of the European Union

Rest API	Ruby 1.9.3p484, Sinatra 1.0

Table *6:* NETMODE monitoring Rest API

Components	Technologies
Monitoring Data Database	PostgreSQL >= 10
Rest API	PostgREST v0.4

D3.01 Requirements and Specifications deliverable describes the functional requirements of the Reputation Service. The highlighted functional requirements in Table 7 are achieved by the developments for the first cycle.

Table *7:* Reputation functional requirements

ID	Title	Coverage in First cycle
REP_01	Reputation Service REST interface	The first cycle implementation fulfils this requirement.
REP_02	REST API documentation	The REST API documentation is not complete and it is expected to be ready at the first semester of the second cycle.
REP_03	Reputation computation engine	In this first cycle HRS is developed, and tested in NETMODE and NITOS testbeds
REP_04	Reputation Service access to monitoring and SLA data	The first cycle implementation fulfils this requirement.

2.4 MYSLICE DEVELOPMENTS FOR THE FIRST CYCLE

The new architecture is composed of 5 layers with a clear separation of concerns: Web frontend, APIs (REST/WS), Database, Services with workers and Library (XML-RPC).

UPMC was working on implementing the backend using Python and frontend using JS REACT. As a result, fully functional web service was implemented including testing suites for the REST calls.

Co-funded by the Horizon 2020 Framework Programme of the European Union

Below table is summarizing REST API calls and status of its implementation:

Table *8:* MySlice REST API Calls

REST call	**Implementation status**
Authentication and Activity	
2.1 User authentication and profile	IMPLEMENTED
2.1.1 POST /login	IMPLEMENTED
2.1.2 GET /usertoken	IMPLEMENTED
2.1.3 POST /usertoken	IMPLEMENTED
2.1.4 GET /profile	IMPLEMENTED
2.2 Activity	
2.2.1 GET /activity/[<id>]	IMPLEMENTED
2.2.2 GET /activity?slice=<id>	IMPLEMENTED
2.2.3 GET /requests/[<id>]	IMPLEMENTED
2.2.4 PUT /requests/<id>	IMPLEMENTED
3 Entities	
3.1 Authorities	
3.1.1 GET /authorities	IMPLEMENTED
3.1.2 GET /authorities/<id>	IMPLEMENTED
3.1.3 GET /users/authorities	IMPLEMENTED
3.1.4 POST /authorities	IMPLEMENTED
3.1.5 PUT /authorities/<id>	IMPLEMENTED
3.1.6 DELETE /authorities/<id>	IMPLEMENTED
3.2 Projects	
3.2.1 GET /projects	IMPLEMENTED

Co-funded by the Horizon 2020 Framework Programme of the European Union

3.2.2 GET /projects/<id>	IMPLEMENTED
3.2.3 GET /authorities/projects	IMPLEMENTED
3.2.4 GET /authorities/<id>/projects	IMPLEMENTED
3.2.5 GET /users/projects	IMPLEMENTED
3.2.6 GET /users/<id>/projects	IMPLEMENTED
3.2.7 POST /projects	IMPLEMENTED
3.2.8 PUT /projects/<id>	IMPLEMENTED
3.2.9 DELETE /projects/<id>	IMPLEMENTED
3.3 Users	
3.3.1 GET /users	IMPLEMENTED
3.3.2 GET /users/<id>	IMPLEMENTED
3.3.3 GET /authorities/users	IMPLEMENTED
3.3.4 GET /authorities/<id>/users	IMPLEMENTED
3.3.5 GET /projects/<id>/users	IMPLEMENTED
3.3.6 GET /slices/<id>/users	IMPLEMENTED
3.3.7 POST /users	IMPLEMENTED
3.3.8 PUT /users/<id>	IMPLEMENTED
3.3.9 DELETE /users/<id>	IMPLEMENTED
3.4 Slices	
3.4.1 GET /slices	IMPLEMENTED
3.4.2 GET /slices/<id>	IMPLEMENTED
3.4.3 GET /slices/<id>/pending	IMPLEMENTED
3.4.4 GET /projects/<id>/slices	IMPLEMENTED
3.4.5 GET /users/slices	IMPLEMENTED

Co-funded by the Horizon 2020
Framework Programme of the European Union

3.4.6 GET /users/<id>/slices?expand=true	IMPLEMENTED
3.4.7 GET /resources/<id>/slices	IMPLEMENTED
3.4.8 POST /slices	IMPLEMENTED
3.4.9 PUT /slices/<id>	IMPLEMENTED
3.4.10 DELETE /slices/<id>	IMPLEMENTED
3.5 Resources	
3.5.1 GET /resources[?timestamp_start=<XXX>&timestamp_end=<XXX>]	IMPLEMENTED
3.5.2 GET /resources/<id>	IMPLEMENTED
3.5.3 GET /slices/<id>/resources	IMPLEMENTED
3.5.4 GET /testbeds/<id>/resources[?timestamp_start=<XXX>&timestamp_end=<XXX>]	IMPLEMENTED
3.5.5 GET /testbeds/<id>/leases?timestamp_start=<XXX>&timestamp_end=<XXX>	IMPLEMENTED
3.5.6 POST /resources	IMPLEMENTED
3.5.7 PUT /resources/<id>	IMPLEMENTED
3.5.8 DELETE /resources/<id>	IMPLEMENTED
3.6 Leases	
3.6.1 GET /leases	IMPLEMENTED
3.6.2 GET /leases[?timestamp_start=<XXX>&timestamp_end=<XXX>]	IMPLEMENTED
3.6.3 GET /testbeds/<id>/leases[?timestamp_start=<XXX>&timestamp_end=<XXX>]	IMPLEMENTED
3.6.4 GET /resources/<id>/leases	IMPLEMENTED
3.6.5 POST /leases	IMPLEMENTED
3.6.6 PUT /leases/<id>	IMPLEMENTED
3.6.7 DELETE /leases/<id>	IMPLEMENTED
3.7 Testbeds	

Co-funded by the Horizon 2020
Framework Programme of the European Union

3.7.1 GET /testbeds	IMPLEMENTED
3.7.2 GET /testbeds/<id>	IMPLEMENTED
3.7.3 POST /testbeds	IMPLEMENTED
3.7.4 PUT /testbeds/<id>	IMPLEMENTED
3.7.5 DELETE /testbeds/<id>	IMPLEMENTED
3.8 Orchestrator	
3.8.1 POST /orchestrator	WORK IN PROGRES

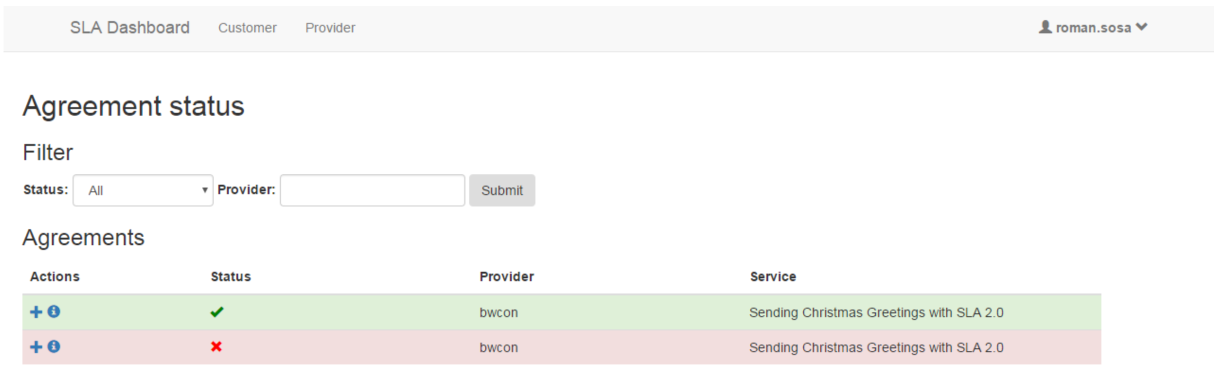
2.5 FRONTEND

The activities of Task 3.1 include the integration of the SLA and Reputation Service with the jFed and MySlice tools of FED4FIRE+. This will be achieved by developing an appropriate plugin for the federation portal (MySlice) and an extension of jFed GUI and CLI. Although these developments are scheduled for the third cycle of the project, some early integrations are already made.

2.5.1 SLA Frontend

As previously mentioned in this document, SLA GUIs have yet to be integrated into MySlice. The pictures below present current independent SLA GUI.

Co-funded by the Horizon 2020 Framework Programme of the European Union



Agreement status

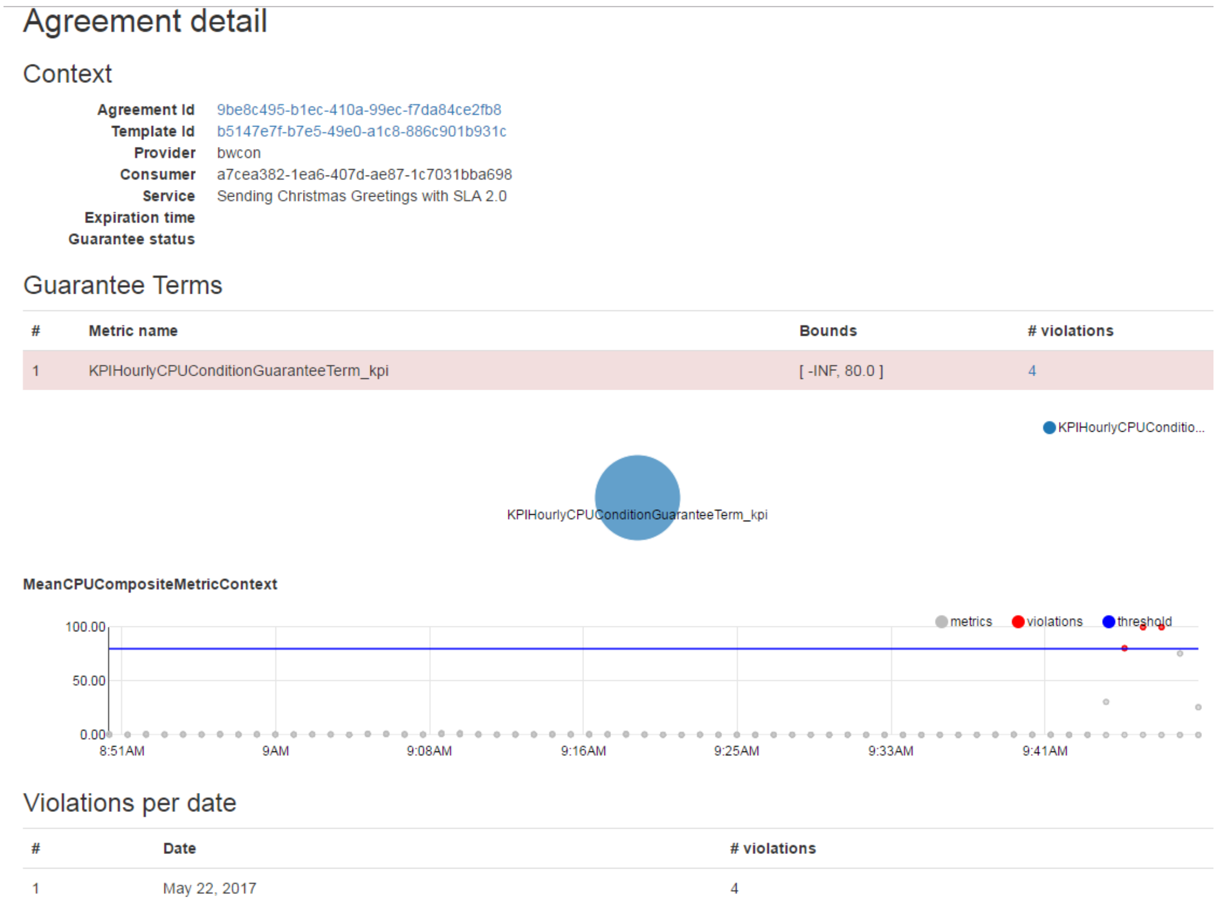
Filter

Status: All Provider: Submit

Agreements

Actions	Status	Provider	Service
+ ℹ	✓	bwcon	Sending Christmas Greetings with SLA 2.0
+ ℹ	✗	bwcon	Sending Christmas Greetings with SLA 2.0

Figure *6:* GUI to check Agreements status



Agreement detail

Context

Agreement Id	9be8c495-b1ec-410a-99ec-f7da84ce2fb8
Template Id	b5147e7f-b7e5-49e0-a1c8-886c901b931c
Provider	bwcon
Consumer	a7cea382-1ea6-407d-ae87-1c7031bba698
Service	Sending Christmas Greetings with SLA 2.0
Expiration time	
Guarantee status	

Guarantee Terms

#	Metric name	Bounds	# violations
1	KPIHourlyCPUConditionGuaranteeTerm_kpi	[-INF, 80.0]	4

● KPIHourlyCPUConditio...
KPIHourlyCPUConditionGuaranteeTerm_kpi
MeanCPUCompositeMetricContext
● metrics ● violations ● threshold
100.00
50.00
0.00
8:51AM
9AM
9:08AM
9:16AM
9:25AM
9:33AM
9:41AM

Violations per date

#	Date	# violations
1	May 22, 2017	4

Figure *7:* Agreement Assessment view

Co-funded by the Horizon 2020
Framework Programme of the European Union

2.6 FUTURE WORK

At the second cycle, the activities of Task 3.1 will focus on integrating the SLA and Reputation Service with the rest FED4FIRE+ testbeds. The testbed providers will determine the QoS and QoE metrics that will be used by the Reputation Engine and the HRS. The testbed providers will be obliged to provide monitoring data for the agreed QoS KPIs. Partners of Task 3.1 will provide detailed guidelines on the development of the SLA service and the appropriate monitoring REST APIs. Our intension is to test and improve the Hybrid Reputation System with many versatile QOS and QoE KPIs.

Next steps and future work in the SLA component relate to pending developments in order to completely cover the requirements for this component. Required technical and functional requirements were defined in D3.01 Requirements and Specifications deliverable. For these, development statuses in first cycle has been provided in section 1.2.2.1 Requirements Coverage in First cycle. Next foreseen developments focus on the enabling subscription to metrics Collection features as well as further elaboration on federated monitoring mechanisms. At level of GUI, the existing SLA independent GUI will be integrated into MySlice new Graphical user interface.

2.7 REFERENCES

[1] Gambetta, D. Can we trust trust?, Trust: Making and Breaking Cooperative Relations, Department of 418 Sociology, University of Oxford, 2000, 213-237.

[2] Kapoukakis, A., Kafetzoglou, S., Androulidakis, G., Papagianni, C. and Papavassiliou, S., Reputation-Based Trust in federated testbeds utilizing user experience. In Proc. of IEEE CAMAD, 2014, pp. 56-60.

[3] Chang, D.Y., Applications of the extent analysis method on fuzzy AHP, European Journal of operational research, 1996, vol. 95, no. 3, pp.649-655.

[4] Kubler, S., Robert, J., Derigent, W., Voisin, A. and Le Traon, Y., A state-of the-art survey & testbed of fuzzy AHP (FAHP) applications," Elsevier Expert Systems with Applications, 2016, vol. 65, pp 398-422.

[5] Patiniotakis, I., Rizou, S., Verginadis, Y. and Mentzas, G., Managing imprecise criteria in cloud service ranking with a fuzzy multi-criteria decision making method," in Proc. of ESOCC, 2013, pp. 34-48.

[6] Garg, S.K., Versteeg, S., and Buyya, R., SMICloud: A Framework for Comparing and Ranking Cloud Services, In Proc. of IEEE UCC, 2011, pp. 210-218.

[7] Coyle, G., The analytic hierarchy process (AHP), Practical strategy: Structured tools and techniques, Pearson Education Ltd, Harlow, UK, 2004, pp.1-11.

3 IMPROVING REPRODUCIBILITY OF EXPERIMENTS – EXPERIMENT-AS-A-SERVICE

3.1 EXPERIMENT SPECIFICATION

The Experiment Specification (ESpec) was developed as a new standard for setting up experiments. It combines various existing industry standards and leverages them to make it easier to fully setup an experiment: from requesting and provisioning the necessary testbed resources to installing software, doing the configuration management and the application deployment. This is also documented towards the users at https://jfed.ilabt.imec.be/espec .

In this way, the ESpec can be used as a base for creating “Experiments-as-a-Service”, where we provide experimenters with fully automated experiments that provide an excellent starting point for doing their scientific research or education activities.

The functionality of this ESpec can also be leveraged to automate continuous testing of the Fed4FIRE+ testbed resources, and the software platforms which have been developed on it. This allows the developers of these platform to detect breaking changes from the moment they happen, which greatly simplifies debugging and decreases the effort needed to sustain these platforms.

The Experiment Specification is not a replacement for the Resource Specification (RSpec) format. Instead, it acts as a bundle (see Figure 8) for an RSpec – which defines the testbed resources that are needed for the experiment – with additional files for the software deployment and configuration. For that second part, we use Ansible: a widely used open source software that automates software provisioning, configuration management and application deployment. As Ansible connects via SSH to the servers it controls and doesn’t need an “agent” to be present on these servers, it is a natural fit for controlling Fed4FIRE+ testbed servers.

The ESpec also provides the necessary glue to make Ansible work: it can generate the necessary configuration files for Ansible, like the inventory-file and an SSH private key for accessing the other servers, and upload them to the Ansible master-node.

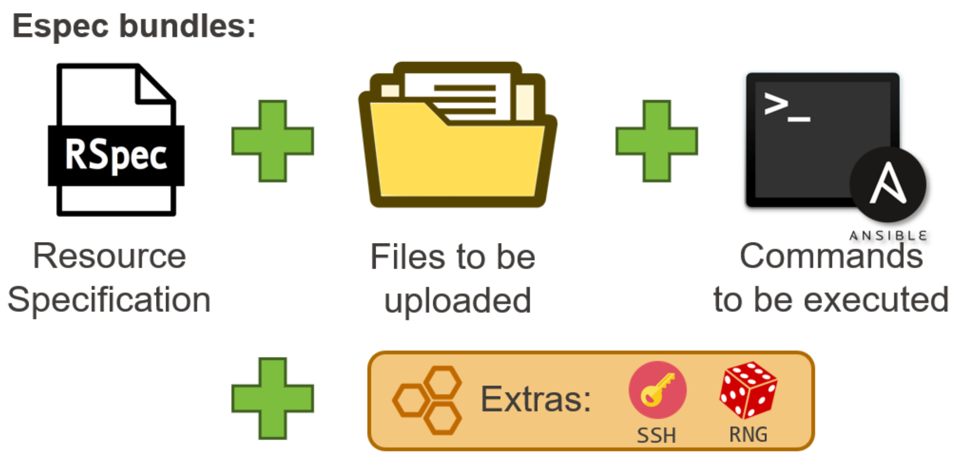


Figure 8: ESpec bundles RSpec, files to be uploaded and scripts

3.2 THE ESPEC BUNDLE

An ESpec bundle is a group of files, which contains:

- `experiment-specification.yml` which contains the meta data that describes what to do with the other files
- An RSpec
- Zero, one or more files to upload
- Zero, one or more scripts to execute

There are different ways to "bundle" the files that form an ESpec:

- Place them in a single directory
- Place them in an archive file (.zip, .tar, .tar.gz, .jar, …)
- Place them in a git repo
- Place them in a github repo

Currently jFed supports all these methods. The `git` and `github` methods are currently not yet supported in the Experimenter GUI, but are supported in the automated tester.

Co-funded by the Horizon 2020 Framework Programme of the European Union

3.2.1 Format of experiment-specification.yml

The ESpec meta data file, `experiment-specification.yml` uses YAML syntax[1], the same markup language as used by Ansible.

A basic file looks like this:

```
version: 1.0-basic
rspec: nodes.rspec
upload: exp-data-files.tar.gz
execute: exp-script.sh
```

Version should for now always be "`1.0-basic`". Future versions will use another identifier.

Note that `execute` will first act as an upload, and then also execute the uploaded file.

Both `upload` and `execute` allow multiple files to be specified. Use a list for that. Example:

```
version: 1.0-basic
rspec: nodes.rspec
upload:
   - exp-files-set1.tar.gz
   - exp-files-set2.zip
execute:
   - setup.sh
   - exp-run.sh
```

Files are uploaded in parallel, but scripts are always executed in order. So in this case, `exp-run.sh` will not run before `setup.sh` has run.

Each time a filename is specified, it is assumed it refers to a bundled file. You can also directly specify the content of the file, or provide an URL to download it from. To do that, the "long" format is used. This also allows extra options such as to which node, and in which path to upload the file.

An example:

```
version: 1.0-basic
```

[1] http://yaml.org/

Co-funded by the Horizon 2020 Framework Programme of the European Union

```
rspec:
   - bundled: 3-nodes.rspec
upload:
   - exp-files-set1.tar.gz
   - bundled: exp-files-set2.tar.gz
     path: /tmp
     nodes: [central, exp1]
   - download: http://example.com/exp-files-set3.tar.gz
   - direct: |
        You can also directly specify the content of a file. This text
will thus be stored on all nodes in /tmp/demo.txt
        Check the yaml syntax of "literal-blocks" for details about syntax
and removing indentation
     path: /tmp/demo.txt
execute:
   - bundled: setup-central-node.sh
     nodes: central
   - bundled: setup-exp-node.sh
     nodes: [exp1, exp2]
   - local: /work/repo/start-exp.sh
     nodes: [exp1, exp2]
```

If no path is specified, the home dir of the user is used. This default can be changed by using dir. `dir` will also create the directories if needed. While this feature can be convenient, keep in mind that permissions of the logged in user are used. So directory creation is typically not allowed everywhere.

You can specify the destination dirs for uploads and scripts (where files in execute are placed) separately. If `scripts` is not specified, it defaults to the same dir as uploads. If uploads is not specified, it defaults to the users home dir.

Paths may start with ~ to indicate they are relative to the users home dir.

An example:

```
version: 1.0-basic
rspec:
   - bundled: example.rspec
dir:
   - path: /tmp/data/
     content: uploads
   - path: ~/scripts/
     content: scripts
   - path: /tmp/extra/
upload:
   - exp-files-set1.tar.gz
   - exp-files-set2.tar.gz
```

Co-funded by the Horizon 2020 Framework Programme of the European Union

```
   - bundled: extra.tgz
     path: /tmp/extra/
execute:
   - setup.sh
   - start.sh
```

In this example, if the users home dir is at `/home/someuser/` the files will be places in:

- `/tmp/extra/extra.tgz`
- `/tmp/data/exp-files-set1.tar.gz`
- `/tmp/data/exp-files-set2.tar.gz`
- `/home/someuser/scripts/setup.sh`
- `/home/someuser/scripts/start.sh`

3.2.2 Dir details

Each dir entry supports the following options:

- `path` (string) *MANDATORY*
- `content` (string)
- `permissions` (string)
- `nodes` (empty, string, or list of string)
- `sudo` (boolean)

`path` specifies the path of the directory used in the ExperimentSpecification. This needs to be an absolute path, or a path relative to the user homedir. Thus, it needs to start with either / or ~.

Non-existing directories will be created (including parent directories). Existing directories will be left as is (unless their permissions are wrong, see later).

`content` is either `upload`, `scripts` or `ansible`, or left unspecified. If specified, files in the referenced sections, for which no path or a relative path is given, will be stored in (or relative to) this dir. If content is unspecified, the dir will just be created if it doesn't exist, which is often useful on its own.

permissions are the required permissions for the directory. If not specified, a default value of `u=rwx` is used. The formats supported by `chmod` are supported, so octal notation (0600) and symbolic notation (`uog=rwx`).

Co-funded by the Horizon 2020
Framework Programme of the European Union

`nodes` is a list empty or unspecified, or one or more nodes on which the dir is created and used as specified in content. If empty, all nodes are used (except for the ansible control machine).

`sudo` if not specified, this option is false. If true, the directory will be created by using `sudo` to gain root privileges. This can be required to create certain directories. Of course, this requires the nodes to have a correctly configured `sudo` command.

3.2.3 File content details

Both `upload` and `execute` need a list with zero, one or more objects that specify what to do. In both cases, these objects contain at least information on the content of the file that is worked with.

To specify file content, the object contains a specific key-value pair. The key determines the method to retrieve the file content, the value depends on the key but typically specifies which specific content to fetch using the method specified in the key. The value is typically a string, but can be an object for certain keys, if they require additional data.

There are different ways to specify which file content to use:

- `bundled`: The file is bundled in the ESpec bundle. The value is the name of the bundles file, possibly including the relative path inside the ESpec bundle. You can also specify a directory from the bundle.
- `download`: The file must be downloaded. The value is the URL of the file.
- `git` or `github`: The file (or dir) is in a git repository. There is currently no difference between specifying git or github. The value is a string with the git URL of the *public* repo. It is also possible to specify more options, using an object as value instead. The following fields are then supported:
    - `url`: The git URL of the repo. May be an HTTP or SSH git url. (mandatory)
    - `branch`: The branch of the repository to use. (optional, default is "master")
    - `dir`: The subdir in the repo to use. If file is specified, this is the base dir of the file. (optional)
    - `file`: The file to fetch from the git repo. If this is not specified, an entire dir, or the entire repo is used. (optional)
    - `username`: The username used for basic authentication. (optional, may not be combined with privateKey)
    - `password`: The password matching the username used for basic authentication. (mandatory if username is specified, forbidden otherwise.)

    - `privateKey`: The private key (in PEM format) needed to access this git repo. Note that the username is specified in the git URL in this case. (optional, may not be combined with username or password)

- `meta`: The file is generated by the client and contains meta data about the experiment/slice. The value is the required meta data. There is currently 1 supported value, which is the name of the metafile which must be generated and uploaded:

    - `manifest.xml`: The manifest RSpec of the slice. If there are multiple AM's involved in this slice then this is the combination of all of their manifest RSpecs.

    - `experiment-info.json`: Information about the experiment in JSON format. This includes info on the user, project, slice, ssh users, and on the nodes.

    - `client_id.txt`: The client_id of the node.

- `generated`: The file must be generated by the client. The value is an object or string. The object should always contain a method field. The string is shorthand for an object with only the method field, with as value the string. There are currently 3 supported methods:

    - `keypair`: This method requires no extra fields. It will generate a random keypair, pass that keypair in the Provision phase, and upload the keypair. This way, all nodes in the experiment can afterwards securely communicate over SSH.

    - `random`: Generate a file with random content. There are 2 extra fields needed: format and length.

        - `format`: Specifies which form the random data has. Options are: password, binary, base64 and alphanum

        - `length`: The length in bytes of the generated random data. Note that for base64 this is the length of the encoded bytes, not the length of the resulting base64 string.

    - `rspec`: Generate an RSpec file. There are 1 extra fields needed: am, and there are some optional fields nodes, prefix and icon.

        - `am`: The component_manager_id of the nodes in the RSpec. You may specify either the URN, the server ID (an integer), or the testbed ID (a string).

        - `nodes`: Either the amount of nodes that need to be generated (an integer), or a list of names for the nodes (a list of strings). Default is 1 (= generate 1 node).

Co-funded by the Horizon 2020 Framework Programme of the European Union

- `prefix`: The prefix used to generate node names, if the nodes option is an integer. The default is "n". (example: For nodes: 2 this would cause 2 nodes to be generated, with names "n1" and "n2".)
- `icon`: The "icon" in the jFed Experimenter GUI to use. If not specified, an icon will be chosen automatically (which is almost always what you need). The name of this icon is the ResourceClass ID that can be found in the fls-api. Examples are: physical-node, wireless, generic-node, vm-xen, vm-openvz, vm, lte, docker-container. For the full list, check the API on https://flsmonitor-api.fed4fire.eu/resourceclass .

An example with generated content:

```
version: 1.0-basic
rspec:
   generated:
       method: rspec
       am: iminds-docker
       nodes:
         - client
         - server
upload:
   - generated: keypair
   - generated:
      method: random
      format: password
      length: 20
     path: random-password.txt
   - generated:
      method: random
      format: binary
      length: 500
     path: /tmp/seed-data.dat
   - git: user@git.example.com:/example-experiment/my-experiment.git
     path: ~/my-exp-repo/
execute:
   - direct: |
       #!/bin/bash -xe

       ~/my-exp-repo/prepare-experiment.sh --set-password ~/random-
password.txt

       ~/my-exp-repo/run-experiment.sh --password ~/random-password.txt --
seedfile /tmp/seed-data.dat

       echo 'All done'
```

Co-funded by the Horizon 2020
Framework Programme of the European Union

For `execute`, `playbook` and `galaxy`, it is allowed not to specify a file content source, but to only specify a path.

In this case, it is assumed the file already exists on the remote node's filesystem at the given path. The file is thus not uploaded to the node, but is already present somehow. Note that is up to you to make sure the file exists. If you want, the file can have different content on each node it is used on.

There are multiple reasons why a file would already be present: it can be included in the diskimage the nodes runs, it can be installed using the install service of the RSpec, or a previous upload or execute step in the Experiment Specification can have created it (directly or indirectly).

An (contrived) example:

```
version: 1.0-basic
rspec: experiment.rspec
upload:
   - bundled: check.sh
     permission: u=rx
execute:
   - path: check.sh
   - path: /usr/bin/sync
ansible:
    host:
       upload:
             - bundled: hello-world-ansible.yml
    playbook:
       - path: hello-world-ansible.yml
```

An example with various git file sources (see this entire example at https://github.com/wvdemeer/espec-test):

```
version: 1.0-basic
rspec: docker.rspec

upload:
    # Fetch and upload a single file from a github repo
    - git:
        url: git@github.com:wvdemeer/espec-test.git
        branch: master
        file: hello3.txt
    # Fetch and upload a file from a subdir of a github repo
    - github:
        url: git@github.com:wvdemeer/espec-test.git
        dir: hello4
```

Co-funded by the Horizon 2020 Framework Programme of the European Union

```
          branch: master
          file: hello4.txt
     # Fetch and upload an entire subdir from a github repo
     - git:
          url: git@github.com:wvdemeer/espec-test.git
          dir: hello56
          branch: master
     - hello78
     - hello9.txt

ansible:
   host:
     type: EXISTING
     name: ansible
     upload: hello2.txt
   playbook: hello-playbook.yml
```

3.2.4 Upload details

Each upload entry supports the following options:

- a content source (*MANDATORY*)
- path (string)
- permissions (string)
- nodes (empty, string, or list of string)

path specifies the path on the *remote* of the file to be uploaded. If no path is specified, the file is uploaded to the upload dir from the `dir` section. Relative paths are relative to the upload directory from the `dir` section.

`permissions` are the required permissions for the uploaded file. If not specified, a default value of u=r is used. The supported format is the same as for dir.

`nodes` is as for dir and specifies the nodes to which the file needs to be uploaded.

Note that for uploading a file, the permissions of the target directory must allow write access to the login user.

3.2.5 Execute details

Each execute entry contains the following options:

- a content source
- `path` (string)

Co-funded by the Horizon 2020 Framework Programme of the European Union

- `pwd` (string)
- `permissions` (string)
- `sudo` (string)
- `log` (string)
- `nodes` (string)

`path` is as with upload, but by default the script dir is used.

`permissions` is as with execute, but the default permissions are u=rx.

`nodes` is as with execute.

`pwd` follows the same rules as `path`, but specifies the working dir in which the script needs to be executed.

`sudo` if not specified, this option is false. If true, the script will be executed using `sudo` to gain root privileges. Of course, this requires the nodes to have a correctly configured `sudo` command.

`log` follows the same rules as path, but specifies where the log file should be written. By default, the same path and base filename as the script will be used, with the extension replaced by '.log'.

- Global configuration Options

There are a few global config options that change the default behavior.

This example shows how to set the config options, and shows the default values (i.e. not providing a config section results in the values in this example being set).

```
version: 1.0-basic
rspec: experiment.rspec
config:
   default_sudo: false                # use sudo when not specified?
   sudo_user: ~                       # null, so no sudo user passed,
which sudo iterprets as root
   default_store_remote_logs: true    # when not overridden, store logs at
the node executing a command?
   default_store_local_logs: true     # when not overridden, download logs
to local client running the ESpec?
   all_nodes_includes_ansible: false  # when uploading or executing on all
nodes, include the ansible node?
execute: test.sh
```

Co-funded by the Horizon 2020
Framework Programme of the European Union

3.2.6 Ansible Support

The Experiment Specification has support for Ansible[2].

Ansible runs on a control machine, which can be:

- The local machine (not yet supported by jFed)
- An existing node in your request RSpec
- And extra node added to your request RSpec

When no preference is specified, a extra node will be added to the request RSpec.

As with `dir`, `upload` and `execute`, a lot of defaults are implied, and if they are used, the syntax can be greatly simplified.

An minimal example experiment-specification file is:

```
version: 1.0-basic
rspec: experiment.rspec
ansible: hello-world-ansible.yml
```

In this example:

- An extra ansible node is added to the RSpec
- Ansible is installed on that node
- The ansible inventory file (and related files) are generated and put on the node
- The specified (bundled) playbook is copied to the node
- ansible-playbook is called to execute the playbook

Off course, one can specify multiple playbooks to execute, and ansible options. The following example is equivalent, but specifies the playbook as a single item in a list.

```
version: 1.0-basic
rspec: experiment.rspec
```

[2] http://www.ansible.com

Co-funded by the Horizon 2020 Framework Programme of the European Union

```
ansible:
    - hello-world-ansible.yml
```

The 2 previous examples are short for the full syntax:

```
version: 1.0-basic
rspec: experiment.rspec
ansible:
    playbook: hello-world-ansible.yml
```

Or as a list:

```
version: 1.0-basic
rspec: experiment.rspec
ansible:
    playbook:
       - hello-world-ansible.yml
```

When all implied defaults are explicitly specified, the same example becomes:

```
version: 1.0-basic
rspec: experiment.rspec
dir:
   - path: ~/ansible/
     content: ansible
ansible:
    host:
       type: ADD
       name: ansible
       galaxy-command: ansible-galaxy
       playbook-command: ansible-playbook
       upload:
          - generated: keypair
       execute:
          - download: https://raw.githubusercontent.com/imec-
ilabt/ansible-init-script/master/install-ansible.sh
    galaxy: ~
    playbook:
       - bundled: hello-world-ansible.yml
         debug: 0
         extra-vars: ~
         extra-vars-from: ~
         log: playbook-exec-hello-world-ansible.log
```

Co-funded by the Horizon 2020
Framework Programme of the European Union

This form explicitly shows most of what adding ansible does. The only thing that is not visible here, is that the ansible inventory file and related files are copied to the ansible dir (`~/ansible/` by default).

Not that you can add any upload and execute steps to the ansible host. Rules to take into account when doing so are:

- The keypair will always be uploaded, even if you do not specify it.
- The default automatic ansible install script will NOT be used if you specify any custom execute step.
- When no path is specified, the files are copied to the ansible dir (`~/ansible/` by default).

The typical use of these upload and execute steps are to copy files needed by the ansible playbooks, and to install ansible on the node.

The ansible host type, can have 3 values:

- `ADD`: A node with the name specified in the name field will be added to the RSpec and used as ansible control machine.
- `EXISTING`: A node already present in the RSpec, whose name matches the name field, will be used as ansible control machine.
- `LOCAL`: The local host will be used as ansible control machine. The name field should not be specified, and the execute and upload steps are not allowed. (not yet supported by jFed)

Some things will not run on the ansible control machine:

- The ansible control machine is the only node that will perform the upload and execute steps specified inside ansible. The nodes list of these steps is ignored and should not be specified.
- The global upload and execute steps that should run on all nodes (no nodes field specified) will NOT run on the ansible control machine. Only if the ansible control machine is explicitly mentioned in a node list, it will be included.

The playbook options allows specifying one or more playbooks. Each of these requires a source, but also allows additional option: `debug`, `extra-vars` and `extra-vars-from`:

- The `debug` option for ansible playbooks is used to determine the number of “v” arguments added to the ansible command. Ansible adds extra debug info for each “v” argument. You can add 0, 1, 2 or 3 “v” arguments.

Co-funded by the Horizon 2020
Framework Programme of the European Union

- The `extra-vars` options takes either a string, or a submap. In the case a string is given, this is passed directly to the ansible-playbook --extra-vars option. A map is converted to json and sotred to a file, that is passed using the same option, but with the "`@file`" syntax: `--extra-vars '@filename'`

- The `extra-vars-from` option also uses the "`@file`" syntax but allows any source.

An example for the extra-vars options:

```
playbook:
       - bundled: hello-world1-ansible.yml
         extra-vars: myvar=hello myothervar=hello2
       - bundled: hello-world2-ansible.yml
         extra-vars:
            myvar: hello
            myothervar: hello2
       - bundled: hello-world3-ansible.yml
         extra-vars-from: extraVars.json
       - bundled: hello-world4-ansible.yml
         extra-vars-from:
            download: http://example.com/my_extra_vars.yml
```

The `galaxy` option is similar to the playbook option, but will run before the playbook option, and will call `ansible-galaxy` to install the requested ansible requirement files. This is used to install ansible modules that add extra features to ansible. In the example above, the galaxy option is null, and thus empty. It expects a list of files like `execute` and `playbook`.

The group options allows specifying a list of groups for the ansible inventory file (sometimes called "ansible hosts file"), and the client IDs of the nodes that belong to them. If a group is defined in both the RSpec and the ESpec, the ESpec overrides the group. Otherwise, both sets of groups are added.

A more complex ansible example, with many defaults overridden:

```
version: 1.0-basic
rspec: my-experiment.rspec
dir:
   - path: /work/ansible/
     content: ansible
ansible:
    host:
       type: EXISTING
       name: control
       galaxy-command: /usr/local/bin/ansible-galaxy
       playbook-command: /usr/local/bin/ansible-playbook
       execute:
```

Co-funded by the Horizon 2020 Framework Programme of the European Union

```
        - my-custom-ansible-install.sh
  galaxy:
     - download: http://example.com/ansible-requirements.yml
     - my-ansible-requirements.yml
  playbook:
     - bundled: setup-software.yml
       debug: 2
     - run-1st-experiment.yml
     - run-2nd-experiment.yml
  group:
     servers:
       - server1
       - server2
     clients:
       - client1
       - client2
```

3.3 USING AN ESPEC IN JFED

The jFed Experiment GUI supports executing ESpecs and gives the experimenter feedback on the actions which are being performed.

Besides the Experimenter GUI, support has also been implemented in the jFed Automated tester, which is used by the Fed4FIRE+ federation monitor. Both the `GuiLogicTest` and `ESpecTest` include ESpec support. This allows the Federation Monitor to do extensive tests, in which

3.3.1 Using an ESpec in the jFed Experimenter GUI

To start an ESpec, the Experimenter clicks the 'Open Espec' button in the 'General' ribbon-tab of the jFed Experimenter GUI:

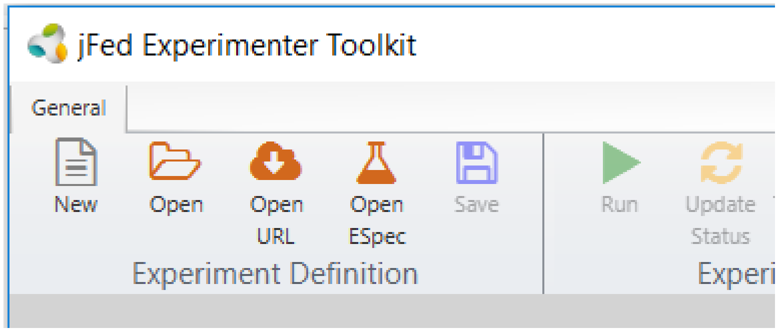
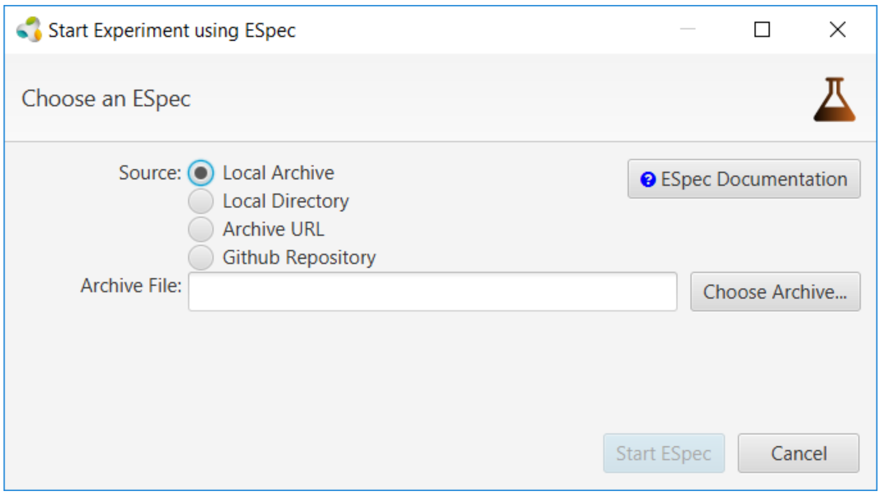


Figure 9: The 'Open Espec' button

This opens a dialog in which the experimenter can specify the location of the ESpec. Depending on the source type, different configuration fields are shown:

Co-funded by the Horizon 2020 Framework Programme of the European Union

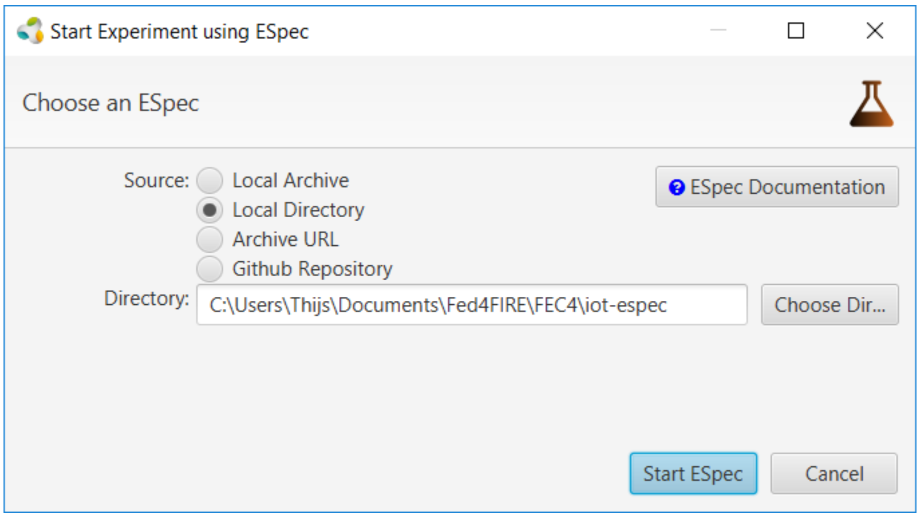


Source: (● Local Archive) Local Directory / Archive URL / Github Repository

Archive File:

Figure 10: Opening an ESpec from a local archive file

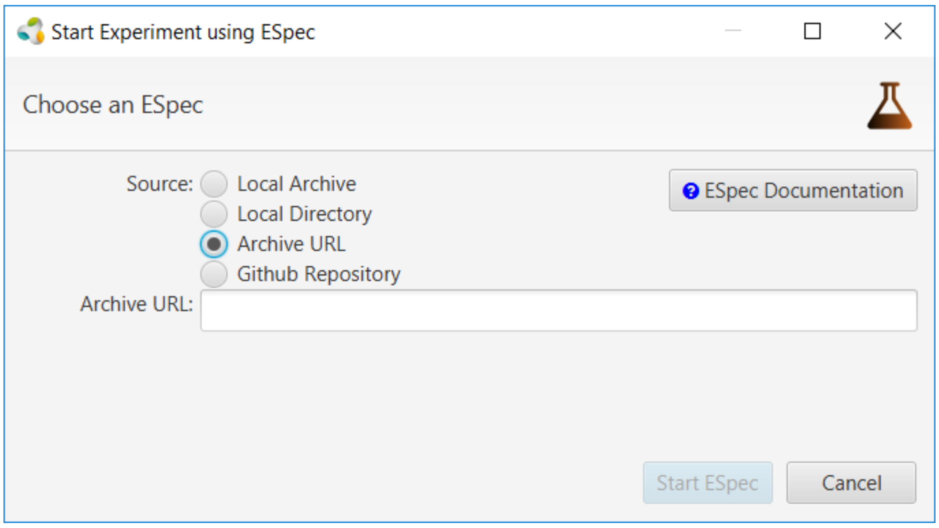
Co-funded by the Horizon 2020
Framework Programme of the European Union



Source: ( ) Local Archive [✓ Local Directory] ( ) Archive URL ( ) Github Repository

Directory: C:\Users\Thijs\Documents\Fed4FIRE\FEC4\iot-espec

Figure 11: Opening an ESpec from a local directory

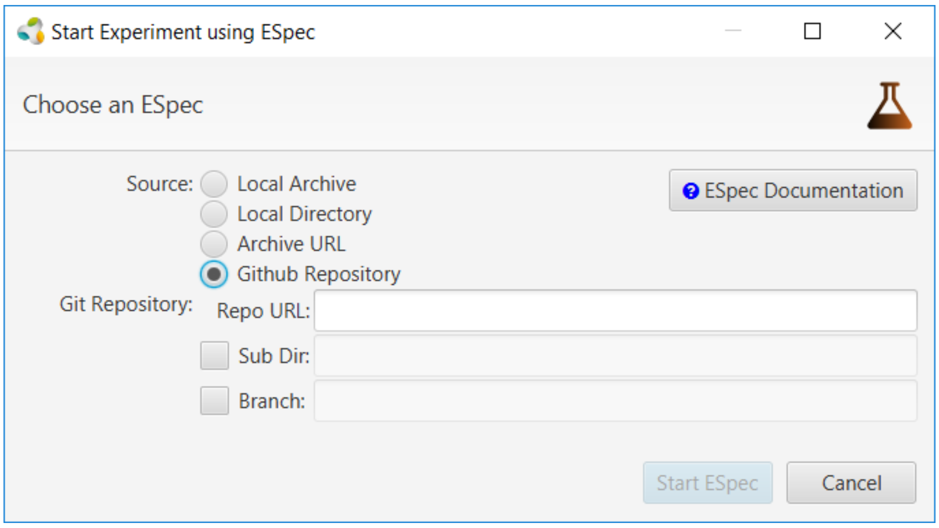


Source: ( ) Local Archive ( ) Local Directory [✓ Archive URL] ( ) Github Repository

Archive URL:

Figure 12: Opening an ESpec from an external archive

Co-funded by the Horizon 2020
Framework Programme of the European Union


Start Experiment using ESpec

Choose an ESpec

Source: Local Archive / Local Directory / Archive URL / [✓ Github Repository]

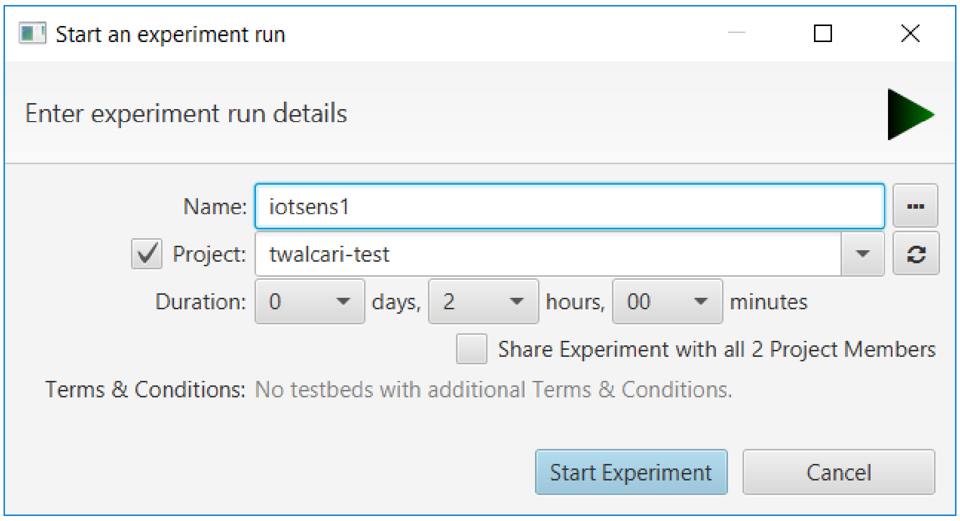
Git Repository: Repo URL:

Sub Dir:

Branch:

Figure 13: Opening an ESpec from a Git(Hub) repository

After specifying a valid source for the ESpec, the experimenter needs to enter a name for the experiment and select the project in which the experiment can be run:


Start an experiment run

Enter experiment run details

Name: iotsens1

[✓] Project: twalcari-test

Duration: 0 days, 2 hours, 00 minutes

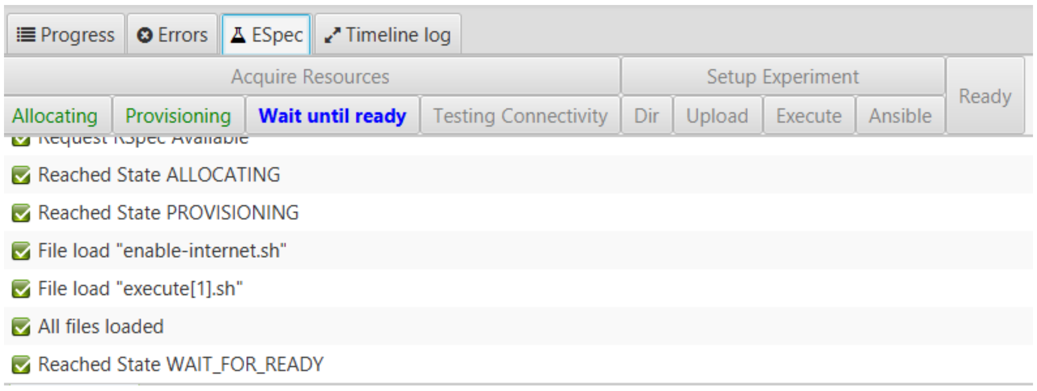
Share Experiment with all 2 Project Members

Terms & Conditions: No testbeds with additional Terms & Conditions.

Figure 14: Experiment Start dialog

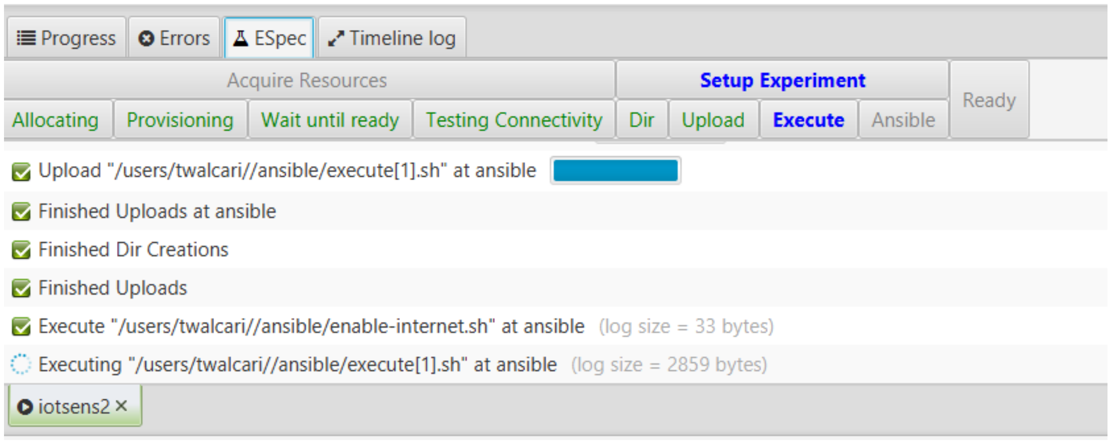
The jFed Experimenter GUI will expose an extra ‘ESpec’-tab which shows the current status of the Experiment Specification.

Co-funded by the Horizon 2020 Framework Programme of the European Union



Request RSpec Available
Reached State ALLOCATING
Reached State PROVISIONING
File load "enable-internet.sh"
File load "execute[1].sh"
All files loaded
Reached State WAIT_FOR_READY

Figure 15: Experiment Specification status, waiting for the testbed resources in this experiment

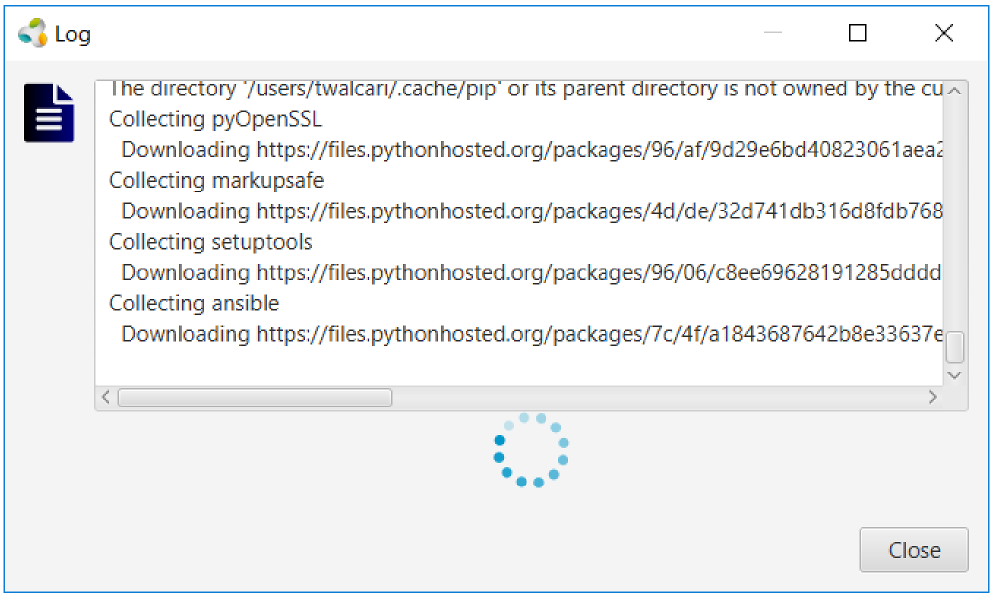


Upload "/users/twalcari//ansible/execute[1].sh" at ansible
Finished Uploads at ansible
Finished Dir Creations
Finished Uploads
Execute "/users/twalcari//ansible/enable-internet.sh" at ansible (log size = 33 bytes)
Executing "/users/twalcari//ansible/execute[1].sh" at ansible (log size = 2859 bytes)

Figure 16: Experiment Specification status, executing a script

Some steps can be clicked on to expose even more detailed information:

Co-funded by the Horizon 2020
Framework Programme of the European Union



```
The directory '/users/twalcari/.cache/pip' or its parent directory is not owned by the cu
Collecting pyOpenSSL
  Downloading https://files.pythonhosted.org/packages/96/af/9d29e6bd40823061aea2
Collecting markupsafe
  Downloading https://files.pythonhosted.org/packages/4d/de/32d741db316d8fdb768
Collecting setuptools
  Downloading https://files.pythonhosted.org/packages/96/06/c8ee69628191285dddd
Collecting ansible
  Downloading https://files.pythonhosted.org/packages/7c/4f/a1843687642b8e33637e
```

Figure 17: Output of the script being executed

Where applicable, there is also a context menu available for a step (reachable by right-clicking on it), which allows the step and all steps before or after to be re-executed. This feature is very useful when creating and debugging an ESpec specification.

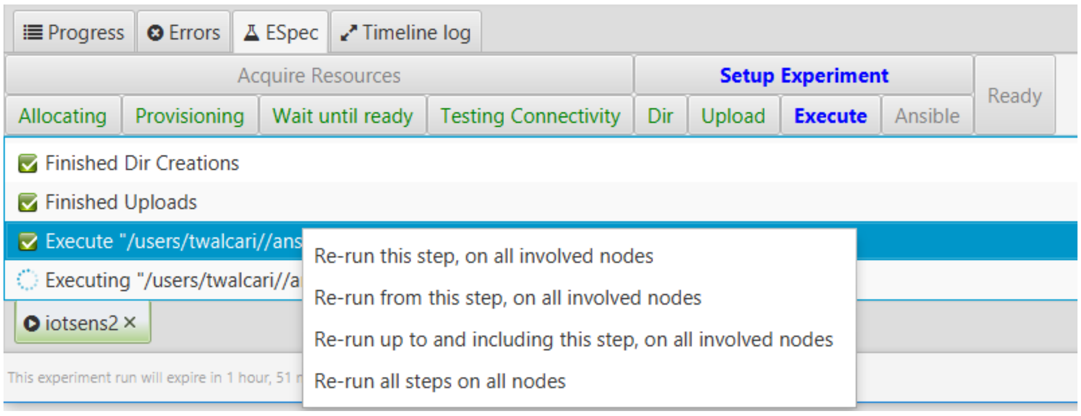
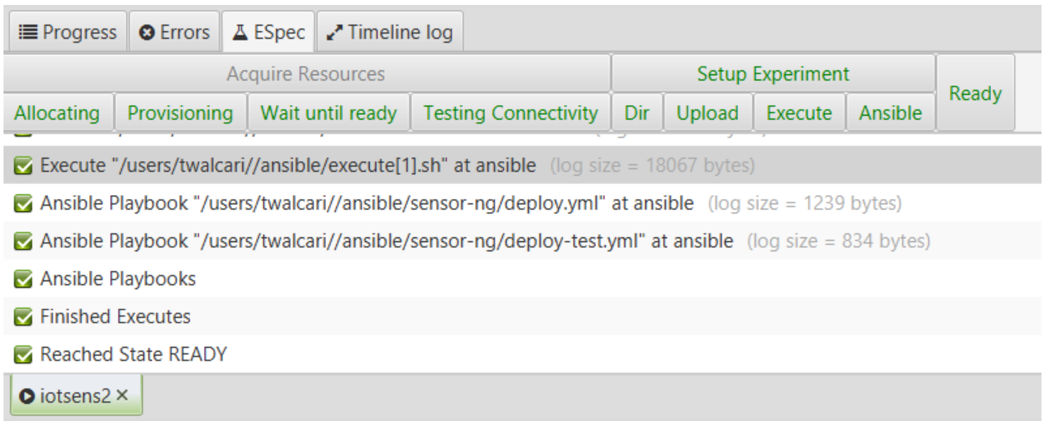

Progress
Errors
ESpec
Timeline log
Acquire Resources
Setup Experiment
Ready
Allocating
Provisioning
Wait until ready
Testing Connectivity
Dir
Upload
Execute
Ansible
Finished Dir Creations
Finished Uploads
Execute "/users/twalcari//ans
Executing "/users/twalcari//a
iotsens2
This experiment run will expire in 1 hour, 51 r
Re-run this step, on all involved nodes
Re-run from this step, on all involved nodes
Re-run up to and including this step, on all involved nodes
Re-run all steps on all nodes

Figure 18: Context menu of a 'step' in the Experiment Specification execution

The overall progress of the Experiment Specification execution can be followed in the progress-buttons shown in the top of the tab. A button with gray text signifies a step which hasn't been executed yet. A button with bold blue text is a currently active step, and a button with green text has been successfully executed.

Co-funded by the Horizon 2020 Framework Programme of the European Union



✓ Execute "/users/twalcari//ansible/execute[1].sh" at ansible (log size = 18067 bytes)
✓ Ansible Playbook "/users/twalcari//ansible/sensor-ng/deploy.yml" at ansible (log size = 1239 bytes)
✓ Ansible Playbook "/users/twalcari//ansible/sensor-ng/deploy-test.yml" at ansible (log size = 834 bytes)
✓ Ansible Playbooks
✓ Finished Executes
✓ Reached State READY

Figure 19: A successfully executed Experiment Specification

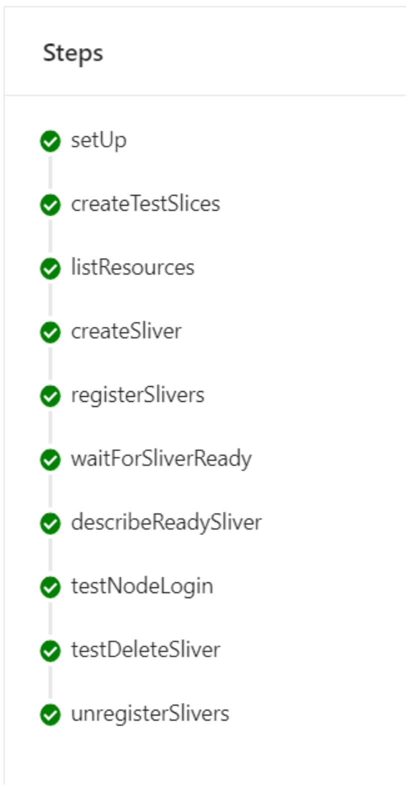
3.3.2 ESpec usage in the Federation Monitor

The ESpec is also used in the federation monitor[3], where it greatly improves the testing capabilities of software deployments. It allows for the complete testing of complex software frameworks, as long as they can be deployed with Ansible.

Previously, the most extensive tests in the federation monitor were the ‘Login test’. This test would execute the full lifecycle of an experiment: create a new slice, list the available resources of a testbed, allocate and provision one of these resources, wait for the resources to become ‘ready’, login on the resources with SSH and do a simple functionality test (eg. a few simple Linux shell-commands) and finally releasing the testbed resource.

[3] https://fedmon.fed4fire.eu

Co-funded by the Horizon 2020 Framework Programme of the European Union


Steps

setUp
createTestSlices
listResources
createSliver
registerSlivers
waitForSliverReady
describeReadySliver
testNodeLogin
testDeleteSliver
unregisterSlivers

Figure 20: Lifecycle steps of a 'Login test'

The ESpec tests extend this scenario in a few ways. All these extensions can be defined in a ‘Federation Monitor Test Definition’, which is a JSON-based definition file.

3.3.2.1 ESpec support in the Fed4FIRE+ Federation Monitor test definitions

Firstly, it is now possible to dynamically specify which testbed resources must be provisioned for the test. The code sample below defines a single testbed-resource where the hardware-type in the Advertisement RSpec is ‘`NUC2014`’ and where the component-id conforms to the regex’es “`nuc0-[0-9]{2}`”, “`nuc10-10`”, “`nuc10-11`”; and which needs to be provisioned with the disk-image with ID ‘`urn:publicid:IDN+wilab1.ilabt.iminds.be+image+emulab-ops:UBUNTU16-64-wilab`’, and in the ansible-group ‘`sensor`’

```
         "requestRSpec": {
            "config": {
              "nodes": [
                  {
                  "count": 1,
                  "diskImage":
"urn:publicid:IDN+wilab1.ilabt.iminds.be+image+emulab-ops:UBUNTU16-64-
wilab",
                  "exclusive": true,
                  "sliverType": "raw-pc",
```

Co-funded by the Horizon 2020 Framework Programme of the European Union

```
                    "ansibleGroup": "sensor",
                    "clientIdBase": "sensor",
                    "availableFilter": true,
                    "hardwareTypeFilter": [
                      "NUC2014"
                    ],
                    "componentIdSelector": "random",
                    "componentNameAllowRegex": [
                      "nuc0-[0-9]{2}",
                      "nuc10-10",
                      "nuc10-11"
                    ],
                    "componentNameDenyThenAllow": false
                  },
                  ...
                  ]
               ...
               }
```

Secondly, it allows to define where the ESpec must be loaded from. The code sample belows defines an ESpec which must be loaded from the branch ‘fedmon’ from a specific git-repository:

```
  "eSpec": {
            "source": "PROVIDE_GIT_REPO_DIR",
            "providedContentSource": "git
git@gitlab.ilabt.imec.be:twalcarius/finterop-sdk.git / fedmon"
          },
```

And lastly, it also supports an additional Ansible playbook to be defined which can be used to test if the deployment of the software stack with the ESpec was successful. To allow the test results to be interpreted automatically, the federation monitor supports extracting text and/or JSON-fragments from the output of this Ansible-playbook, and saving this alongside the test results. This allows for a far more granular result than just looking to the exit status of the `ansible-playbook` command.

The code fragment below shows the definition of such an Ansible-test, where the output between delimiters ‘`====coapthon-cli-vs-californium-server====`’ is parsed as a JSON and saved with the testresults.

```
      "ansibleTests": [
        {
          "debug": false,
          "enabled": true,
```

Co-funded by the Horizon 2020 Framework Programme of the European Union

```
          "extract": [
            {
              "name": "coapthon-cli-vs-californium-server",
              "type": "JSON",
              "delim": "====coapthon-cli-vs-californium-server====",
              "scope": "TESTINSTANCE"
            },
            ...
          ],
          "playbookExe": "/usr/bin/ansible-playbook",
          "playbookUrl": "git
git@gitlab.ilabt.imec.be:twalcarius/finterop-sdk.git / fedmon",
          "failureRegex": "(failed=[^0]|unreachable=[^0])",
          "timeoutInSec": 2400,
          "playbookFileNameInArchive": " tests/test.yml"
        }
      ],
```

A full ‘Federation Monitor Test Definition’ using an ESpec looks as follows:

```
{
  "id": 1413,
  "name": "F-Interop IoT tests",
  "testDefinition": "https://flsmonitor-
api.fed4fire.eu/testdefinition/nodelogin2long",
  "testVersion": "long",
  "enabled": true,
  "frequency": "https://flsmonitor-api.fed4fire.eu/frequency/22",
  "parameters": {
    "method_required_for_success": "ansible",
    "server": "https://flsmonitor-api.fed4fire.eu/server/484",
    "user": "https://flsmonitor-api.fed4fire.eu/user/ftester",
    "automated_tester_config": {
      "resources": [
        {
          "eSpec": {
            "source": "PROVIDE_GIT_REPO_DIR",
            "providedContentSource": "git
git@gitlab.ilabt.imec.be:twalcarius/finterop-sdk.git / fedmon"
          },
          "slice": {
            "expireTimeMin": 200
          },
          "requestRSpec": {
            "config": {
              "nodes": [
                {
                  "count": 1,
                  "serverId": 311,
                  "diskImage":
"urn:publicid:IDN+wall2.ilabt.iminds.be+image+emulab-ops:UBUNTU16-64-STD",
                  "exclusive": true,
                  "sliverType": "raw-pc",
                  "ansibleGroup": "server",
                  "clientIdBase": "server",
                  "availableFilter": true,
                  "componentIdSelector": "noassign"
                },
                {
                  "count": 1,
                  "serverId": 311,
                  "diskImage":
"urn:publicid:IDN+wall2.ilabt.iminds.be+image+emulab-ops:UBUNTU16-64-STD",
                  "exclusive": true,
                  "sliverType": "raw-pc",
                  "ansibleGroup": "ansible",
                  "clientIdBase": "ansible",
                  "availableFilter": true,
                  "componentIdSelector": "noassign"
                },
                {
                  "count": 1,
                  "diskImage":
"urn:publicid:IDN+wilab1.ilabt.iminds.be+image+emulab-ops:UBUNTU16-64-
wilab",
                  "exclusive": true,
                  "sliverType": "raw-pc",
                  "ansibleGroup": "sensor",
                  "clientIdBase": "sensor",
                  "availableFilter": true,
                  "hardwareTypeFilter": [
                    "NUC2014"
                  ],
                  "componentIdSelector": "random",
                  "componentNameAllowRegex": [
                    "nuc0-[0-9]{2}",
                    "nuc10-10",
                    "nuc10-11"
                  ],
                  "componentNameDenyThenAllow": false
                }
              ]
            },
```

Co-funded by the Horizon 2020
Framework Programme of the European Union

```
          "source": "GENERATE_USING_ADVERTISEMENT"
        },
        "waitForReady": {
          "maxTimeMin": 100
        },
        "overrideESpecRSpec": true
      }
    ],
    "ansibleTests": [
      {
        "debug": false,
        "enabled": true,
        "extract": [
          {
            "name": "coapthon-cli-vs-californium-server",
            "type": "JSON",
            "delim": "====coapthon-cli-vs-californium-server====",
            "scope": "TESTINSTANCE"
          },
          {
            "name": "californium-cli-vs-californium-server",
            "type": "JSON",
            "delim": "====californium-cli-vs-californium-server====",
            "scope": "TESTINSTANCE"
          },
          {
            "name": "californium-cli-vs-coapthon-server",
            "type": "JSON",
            "delim": "====californium-cli-vs-coapthon-server====",
            "scope": "TESTINSTANCE"
          },
          {
            "name": "coapthon-cli-vs-coapthon-server",
            "type": "JSON",
            "delim": "====coapthon-cli-vs-coapthon-server====",
            "scope": "TESTINSTANCE"
          },
          {
            "name": "californium-cli-vs-contiki-server",
            "type": "JSON",
            "delim": "====californium-cli-vs-contiki-server====",
            "scope": "TESTINSTANCE"
          },
          {
            "name": "californium-cli-vs-contiki-ng-server",
            "type": "JSON",
            "delim": "====californium-cli-vs-contiki-ng-server====",
            "scope": "TESTINSTANCE"
```

Co-funded by the Horizon 2020
Framework Programme of the European Union

```
            }
          ],
          "playbookExe": "/usr/bin/ansible-playbook",
          "playbookUrl": "git
git@gitlab.ilabt.imec.be:twalcarius/finterop-sdk.git / fedmon",
          "failureRegex": "(failed=[^0]|unreachable=[^0])",
          "timeoutInSec": 2400,
          "playbookFileNameInArchive": " tests/test.yml"
        }
      ],
      "nodeLoginTest": {
        "enabled": true
      }
    }
  },
  "selfTestImmune": false,
  "@id": "https://flsmonitor-api.fed4fire.eu/testinstance/1413",
  "@type": "TestInstance"
}
```

3.3.2.2 Execution of Federation monitor tests with ESpecs

The results of a test execution with the extensions described above contains some extra information.

Firstly, it details which nodes were selected dynamically:

```
rspec-request-fixed-nodes: [
      "urn:publicid:IDN+wilab1.ilabt.iminds.be+node+nuc10-11"
],
```

Secondly, the extracted info from the Ansible test-output is also merged into the test result.

When the output of the Ansible playbook contains something like:

```
[fulltest.ui_stub]:====coapthon-cli-vs-californium-server==== {\n
\"testname\": \"coapthon-cli-vs-californium-server\",\n
\"_api_version\": \"1.1.0\",\n    \"tc_results\": [\n        {\n
\"description\": \"No interoperability error was detected,\",\n
\"testcase_id\": \"TD_COAP_CORE_01\",\n            \"verdict\":
\"pass\",\n            \"partial_verdicts\": [\n               [\n
\"TD_COAP_CORE_01_step_02\",\n                    null,\n
\"CHECK step: postponed\",\n                    \"\"\n                ],\n
[\n                    \"TD_COAP_CORE_01_step_03\",\n
```

```
null,\n                    \"CHECK step: postponed\",\n
\"\"\n                ],\n                [\n
...
====coapthon-cli-vs-californium-server====
```

Then the content between the two delimiters will have been parsed as JSON, and be available in the test result as follows:

```
extract: {
      coapthon-cli-vs-californium-server: {
            testname: "coapthon-cli-vs-californium-server",
            tc_results: [
            {
                  verdict: "pass",
                  description: "No interoperability error was detected,",
                  testcase_id: "TD_COAP_CORE_01",
                  ...
            }
            ...
      }
      ...
}
```

3.3.2.3 Visualization of extracted test results in the Federation Monitor

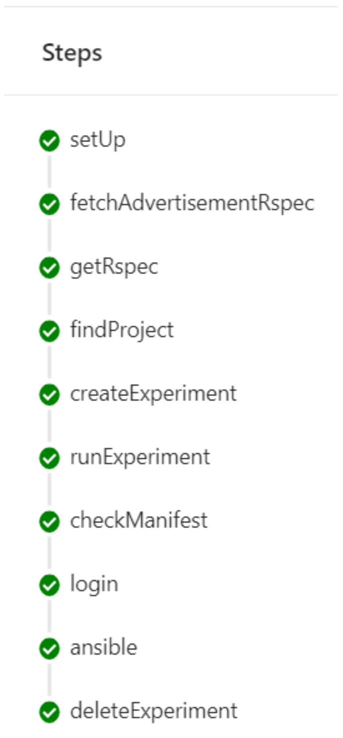
The test steps of an ESpec test in the Federation Monitor are structured differently from a login test. This is because the implementation used by an ESpec test is the same as the one used in the jFed Experimenter GUI.

The step ‘runExperiment’ bundles:

- The allocation and provisining of the resources;
- Waiting for these resources to become ‘ready’;
- Executing the `execute`, `upload` and `ansible`-steps defined in the `experiment-definition.yml`

The step ‘ansible’ runs the Ansible test-playbook to verify the setup.

Co-funded by the Horizon 2020 Framework Programme of the European Union

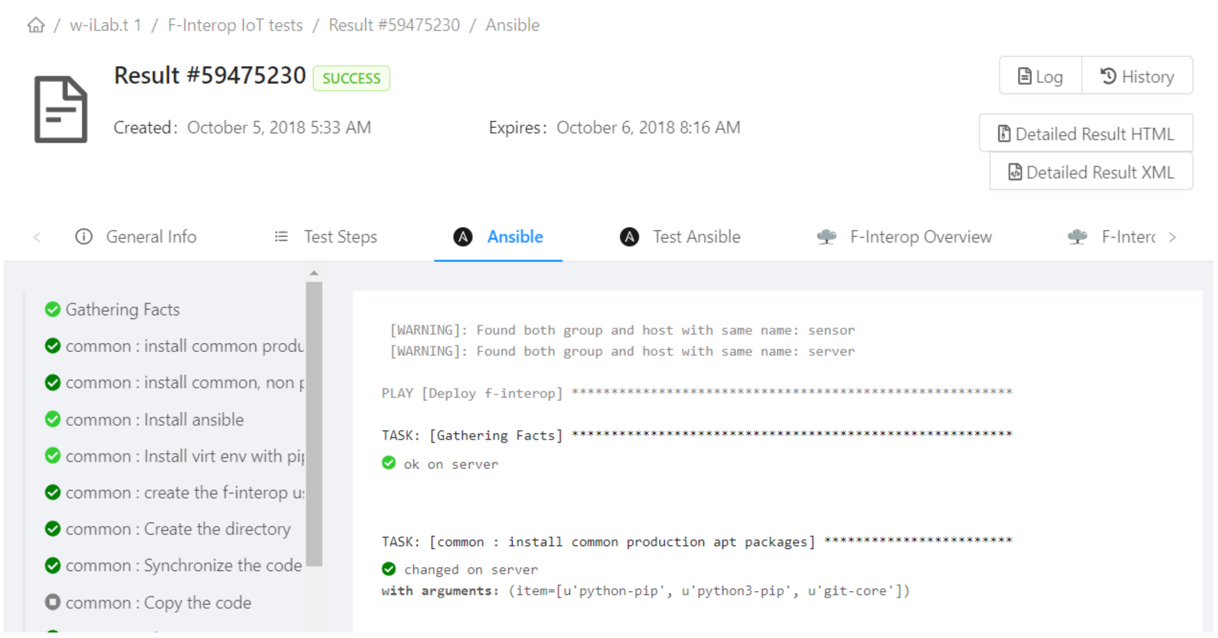

Steps

✓ setUp
✓ fetchAdvertisementRspec
✓ getRspec
✓ findProject
✓ createExperiment
✓ runExperiment
✓ checkManifest
✓ login
✓ ansible
✓ deleteExperiment

Figure 21: Lifecycle steps of an ESpec test

The Federation monitor makes the output of all Ansible playbook executions directly available via tabs on the detailed result-page on the website. This allows for easy verification and debugging of the test results.

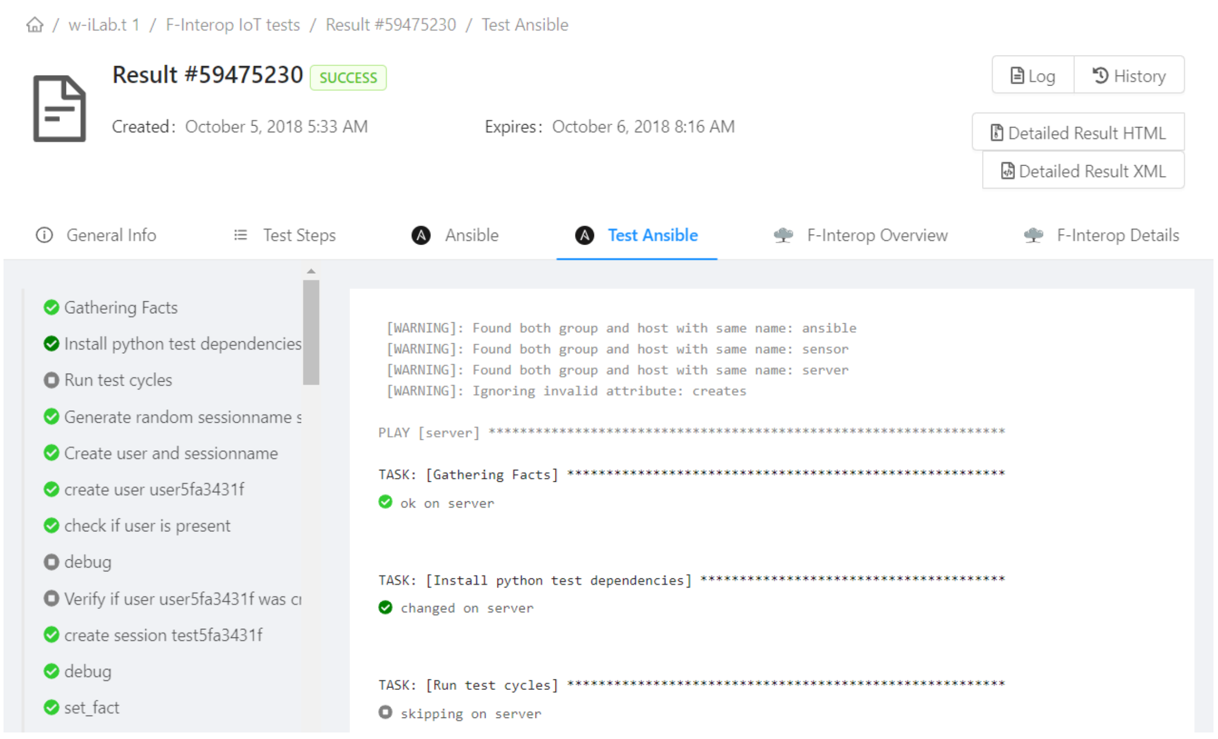
Co-funded by the Horizon 2020
Framework Programme of the European Union



- ✓ Gathering Facts
- ✓ common : install common produ
- ✓ common : install common, non p
- ✓ common : Install ansible
- ✓ common : Install virt env with pi
- ✓ common : create the f-interop u
- ✓ common : Create the directory
- ✓ common : Synchronize the code
- ○ common : Copy the code

```
 [WARNING]: Found both group and host with same name: sensor
 [WARNING]: Found both group and host with same name: server

PLAY [Deploy f-interop] ********************************************************

TASK: [Gathering Facts] ********************************************************
✓ ok on server


TASK: [common : install common production apt packages] ************************
✓ changed on server
with arguments: (item=[u'python-pip', u'python3-pip', u'git-core'])
```

Figure 22: Output from an 'ansible'-command in an ESpec



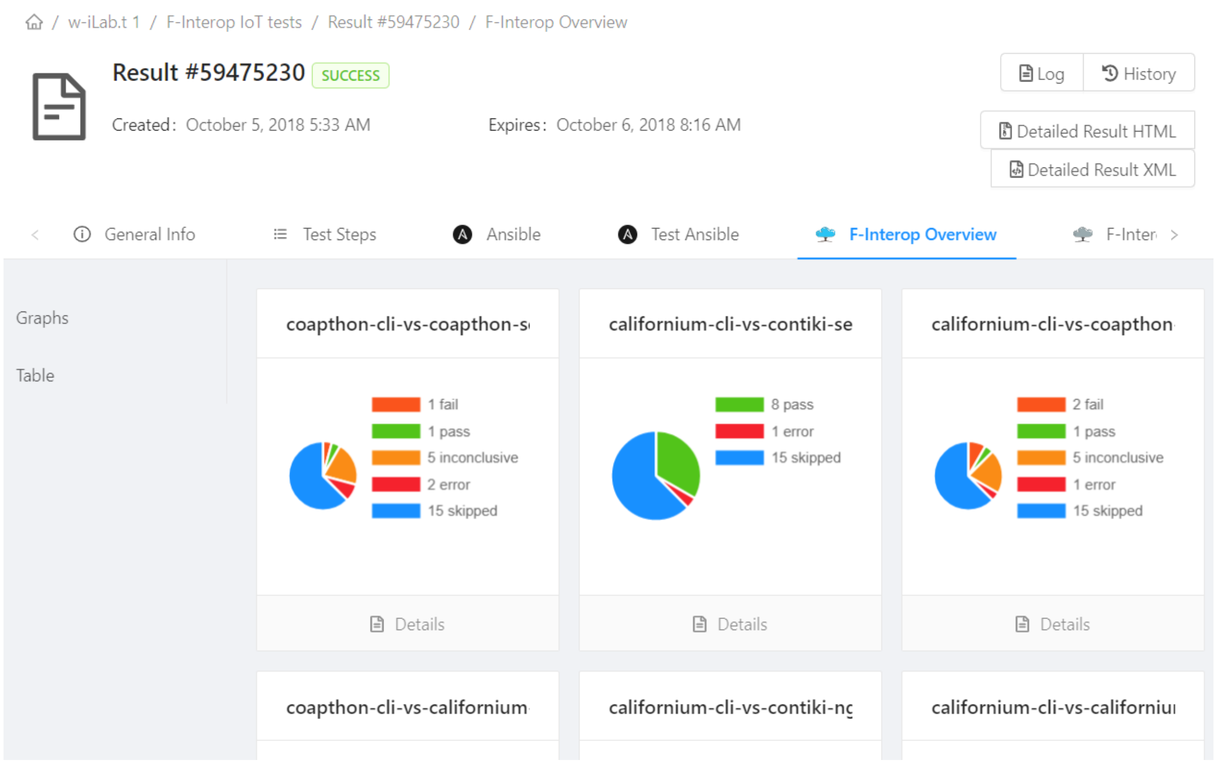
- ✓ Gathering Facts
- ✓ Install python test dependencies
- ○ Run test cycles
- ✓ Generate random sessionname s
- ✓ Create user and sessionname
- ✓ create user user5fa3431f
- ✓ check if user is present
- ○ debug
- ○ Verify if user user5fa3431f was c
- ✓ create session test5fa3431f
- ✓ debug
- ✓ set_fact

```
 [WARNING]: Found both group and host with same name: ansible
 [WARNING]: Found both group and host with same name: sensor
 [WARNING]: Found both group and host with same name: server
 [WARNING]: Ignoring invalid attribute: creates

PLAY [server] ******************************************************************

TASK: [Gathering Facts] ********************************************************
✓ ok on server


TASK: [Install python test dependencies] ***************************************
✓ changed on server


TASK: [Run test cycles] ********************************************************
○ skipping on server
```

Figure 23: Output from the Ansible test-playbook

Co-funded by the Horizon 2020
Framework Programme of the European Union

Where useful, the Federation monitor website also interprets the extracted information from the Ansible test-playbook, and displays this in a concise manner. In the example below the test results of an test of the F-Interop project have been visualised for easy interpretation.


w-iLab.t 1 / F-Interop IoT tests / Result #59475230 / F-Interop Overview

Result #59475230 SUCCESS

Created: October 5, 2018 5:33 AM

Expires: October 6, 2018 8:16 AM

Figure 24: Visualisation of the information extracted from an Ansible test-playbook

Co-funded by the Horizon 2020 Framework Programme of the European Union

4 REPRODUCIBILITY THROUGH CONTINUOUS HARDWARE VERIFICATION

When using testbeds in the context of experimental computer science, the ability to produce trustworthy and reproducible experiments results depends greatly on the trustworthiness of the infrastructure itself. Unfortunately, several factors many issues such as software misconfiguration, hardware heterogeneity, or service failures, can remain undetected and affect the quality of experimental results. This section presents the design and implementation of an automated testbed testing framework. This framework was deployed in the context of the Grid'5000 project, and uncovered more than one hundred of issues.

4.1 INTRODUCTION

Reproducibility has been the focus of a lot of interest over the recent years, in science in general, and in computer science specifically. But most of this focus has been targeted at the *reproducibility of data analysis*, which is usually handled by a pipeline [1] of several steps involving various tools, starting from *measured data* and going up to *figures* and *tables* included in an article. The various steps of that pipeline involve code for pre-processing the measured data, data analysis, and data presentation. However, this focus on *reproducibility of data analysis* ignores the important question of how *measured data* is produced.

Experimental computer science generally involves two main methods to acquire data about *systems under test*: simulation, and experimentation on testbeds. Experimenting on a testbed is a challenging task, and usually involves many different tools: the testbed itself, of course, but also experiment orchestration solutions [2] ranging from shell scripts to complex frameworks, load or failure injectors, emulation solutions, measurement tools, etc. Each of those components has a huge impact on the experiment and the results that will be obtained from it. In theory, experimenters should include a qualification and calibration phase in their experiments, and confirm that this whole stack meets its specification. But unfortunately this is very rarely done in practice, probably due to lack of time or adequate tools.

Still, assessing the correctness of software, e.g. with software testing techniques, is a relatively well-understood process [3]. But the bottom layer of the stack, that is, the testbed itself, raises specific challenges: testing infrastructure is an entirely different story. The complex mix of software and hardware, deployed at scale, provides potential for many difficult-to-detect issues, such as hardware misconfigurations or failures, or software bugs that happen randomly or only at scale.

This section describes work that was carried out in the context of the Grid'5000 testbed in order to systematically test the infrastructure and its services, with the goal of increasing the reliability and the trustworthiness of the testbed by uncovering problems that would otherwise harm the repeatability and the reproducibility of experiments.

The section is structured as follows. Section 4.2 provides the necessary context about the Grid'5000 testbed. Section 4.3 details motivations for this work, and specific challenges that were encountered. Then, Section 4.4 describes the solution that was implemented, before

Co-funded by the Horizon 2020
Framework Programme of the European Union

some results are discussed in Section 4.5. Finally, related work is presented in Section 4.6 before the section is concluded in Section 4.6.

4.2 CONTEXT: THE GRID'5000 TESTBED

This section provides some background information about the Grid'5000 testbed, and about the way it is being operated, in order to support the design choices explained later.

The Grid'5000 project was initiated in 2003 with the goal to provide a testbed to experiment on Grid computing. The focus later moved to become a versatile testbed serving other areas of distributed computing (P2P, HPC, Cloud, Big Data, networking). The testbed itself was open to users in 2005. Each year, the testbed sees about 550 active users (users making at least one resource reservation) that produce about 100 publications.

Grid'5000 is currently composed of 8 sites (Figure 25) located in France and in Luxembourg. Each Grid'5000 site is composed of one or more sets of homogeneous machines called *clusters*. All machines from the same *cluster* are usually bought at the same time. At the time of writing, Grid'5000 has a total of 32 clusters, composed of a total of 894 nodes. At the networking level, all sites are interconnected with a dedicated 10 Gbps backbone network.

Some specific hardware is also available on Grid'5000: various generations of HPC networks (mostly Infiniband), of GPUs, and Xeon Phi co-processors.

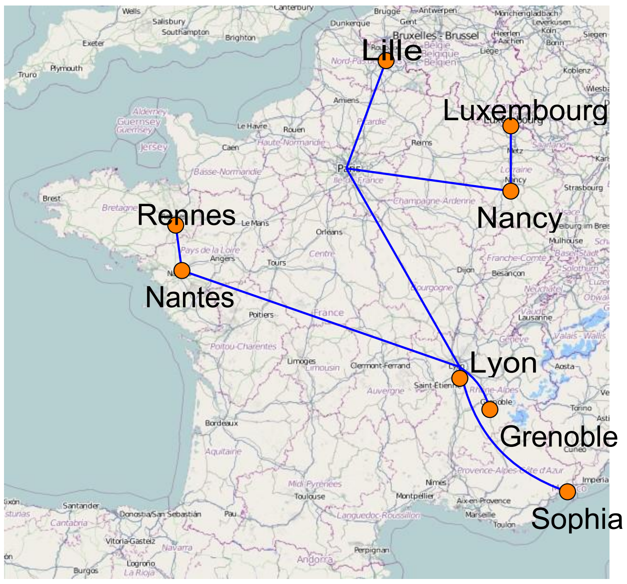


Figure 25:Grid'5000 sites and interconnection network

The main Grid'5000 features and services are:

- *Resources description and discovery*: a REST API (called the Reference API) provides detailed information about all resources in the testbed. Each time a node boots, its description is verified by a tool called *g5kchecks* that collects information using various

Co-funded by the Horizon 2020 Framework Programme of the European Union

inventory tools [4], to ensure that the Reference API contains correct and up-to-date information.

- *Resources reservation*: a resource manager (OAR [5]) is used by users to reserve resources for a specified duration. A rather complex Usage Policy[4] ensures fair sharing of resources between users during the day, while large reservations (generally made in advance) can use all resources at night and during weekends.
- *Nodes reconfiguration*: the testbed provides a default (called *standard*) environment where users do not have root access, similarly to what is available on traditional HPC clusters. On top of that, bare-metal deployment using Kadeploy [6] provides users with the ability to run custom operating system images and get root access. The Grid'5000 team provides a number of images (called *reference environments*), and users can also create custom environments. Out-of-band consoles to nodes are also available to users.
- *Network reconfiguration*: the KaVLAN tool provides the ability to reconfigure the network. Each node's network interface (and nodes have up to four network interfaces) can be put in a different VLAN, which are reserved using OAR. This is typically used for cloud experiments [7] or networking experiments. Additionally, those VLANs can also be propagated inside the backbone links, providing isolation for multi-site experiments.
- *Network and power monitoring*: the Kwapi tool [8] provides an API, a live visualization interface and an archive of measurements of network traffic and power consumption of all nodes of the testbed, captured at high frequency.

While the testbed is distributed, the Grid'5000 engineers work as a single team that manages all sites in a single administrative domain. Each site is still assigned to a specific engineer (mainly in order to build knowledge about local specificities, and to perform maintenance in machine rooms), but the configuration of services is managed centrally, through the use of a configuration management system (Puppet).

4.3 MOTIVATIONS

4.3.1 Very few bugs are reported

Reporting bugs or asking technical questions correctly is a difficult process [9], [10]. Typical users of testbeds (PhD students or post-doctoral students) rarely have that skill, or lack the confidence to report a bug about an infrastructure that they do not fully understand. In the context of Grid'5000, the fact that the team is geographically distributed also makes it hard to talk to a local contact that could confirm the problem informally. As a result, we receive very few bugs from users, despite, as we will show later, a large number of issues that could and should be reported.

Testbed operators would be in a good position to find and report such issues, as they clearly have the expertise about how the testbed should behave. But they often have limited

[4] https://www.grid5000.fr/w/Grid5000:UsagePolicy

Co-funded by the Horizon 2020 Framework Programme of the European Union

experience of *using* the testbed, especially of the large variety of services that are offered to users, and as a result they are unlikely to face all the problems that users encounter.

4.3.2 But many bugs should be reported

Testbeds such as Grid’5000 can produce a large number of different and interesting issues.

A first obvious factor is its scale: 8 sites, 32 clusters and 894 nodes provide plenty of potential for subtle problems. Also, while methods to standardize software configuration at scale are reasonably well understood (e.g. configuration management), hardware is much more difficult to deal with: its configuration sometimes requires manual steps (inside BIOS for example), it tends to fail much more randomly than software (for example, due to aging), and exhibits silent and subtle failure patterns (as a real example: vibrations causing screws attaching hard disk drives to become loose, causing additional vibrations, that can have a performance impact [11]).

Another, less obvious factor is the software stack and the ecosystem of services deployed on the testbed. Some core services are heavily used, but next to them, testbeds are always trying to design new services to address new experimentation needs. When they are first made available to the public, they rarely immediately pick up a strong user base, able to detect issues as soon as they occur. As a result, it is not unusual for a new service to be broken for a long period of time without testbed operators being aware, which is, of course, detrimental to attracting users.

4.3.3 And bugs can have dramatic consequences

In a testbed where most of the users are interested in measuring performance, subtle bugs can have a huge impact. For example, a misconfigured service or node could reduce performance by 5% or 10%, and thus lead experimenters to wrong conclusions about the solutions they are comparing, which could result in the need to retract a paper. Example cases where this could happen, all based on real facts, are:

- Different CPU settings, such as power management (Cstates), hyperthreading, or turbo boost;
- Different disk firmware version;
- Different cache settings in a disk drive;
- Cabling issue that would cause wrong measurements by testbed-managed monitoring services (e.g. measuring the power consumption of another one).

In addition, there are also many problems that can be found at the software side, causing services to be unreliable and making it much harder to automate experiments.

4.4 DESIGN OF OUR TESTBED TESTING FRAMEWORK

In order to design a testbed testing framework, we leveraged the Jenkins automation server to run testing scripts. But we had to work around several limitations of Jenkins through external developments, most notably for tests scheduling, and analyzing and summarizing results. The following paragraphs explore each of those aspects.

Co-funded by the Horizon 2020
Framework Programme of the European Union

Overall, we tried to build on the widely accepted best practices in software engineering about test suites, continuous integration (CI) and continuous delivery (CD), but had to adjust because of specifics of the context, and of our goals.

4.4.1 Jenkins automation server

Jenkins [12] is the de facto standard for automating processes. In a nutshell, it can be seen as the *cron* Unix service on steroids. Using Jenkins, one can define *jobs* (tasks) that are executed in a specified environment, started by various means. The result and output of jobs are stored by Jenkins (as well as historical data).

Jenkins can be extended through plugins. A central plugin for our work was the Matrix Project Plugin, that adds support for defining jobs as matrices of several options. We used that for most tests in order to cover all possible configurations. For example, the test_environments job is in charge of testing the deployment of each of the 14 reference environments provided by the technical team, on each of the 32 clusters, resulting on 448 configurations for that test alone. Another related useful plugin was the Matrix Reloaded Plugin, that adds support for restarting a subset of configurations in a Matrix job.

Several other plugins also proved useful, such as the Build Timeout Plugin (to work around unexpected problems in some test scripts).

However, Jenkins alone proved insufficient for our needs, mainly for two aspects: fine-grained job scheduling, and analyzing and summarizing results. How we addressed those is detailed in the following sections.

4.4.2 Job scheduling

The scheduling of test jobs on the testbed was difficult to design (and a challenge that caused a previous iteration of work on this topic to fail). The requirements are fairly complex. First, different kinds of tests need to be addressed: some that focus on software, and only require one node per cluster; other that focus on hardware, and require all nodes from a cluster. Second, the scheduling must handle the fact that the resources might not be available (because of resources already reserved, or resources reservations made in advance for the future, that will prevent the allocation of resources for the required duration). Third, the scheduling must avoid disrupting other usages of the testbed.

Submitting test jobs as normal resources reservations, and waiting until resources are allocated, is not an option because: (1) Jenkins has a limited number of *workers* (job slots), and a pending reservation would use such for slot, possibly for days; (2) The test jobs would compete with usermade resources reservations, and possibly block resources when users would want to use them.

Also, submitting jobs at the same time every day or week is not an option either, because it would be unlikely that the resources would be available.

The solution that was implemented was a tool external to Jenkins that would, for each configuration of each test job, and on a regular basis (every 10 minutes): (1) query the status of the configuration, and make a decision about whether a new run should be attempted, based on the current state of the test (successful, failed) and on a per-job delay between attempts;

Co-funded by the Horizon 2020 Framework Programme of the European Union

(2) evaluate the status of the testbed to determine, using a basic analysis, if the job could start immediately; (3) start the job, and if it ends up not being scheduled by the testbed's resource manager after a few minutes (due to conditions that were not considered in step 2, cancel it (and mark it as *unstable* in Jenkins).

As a result, test jobs are only scheduled when resources are available, and as frequently as possible.

To further reduce the impact on users, two additional rules have been implemented. First, the scheduler avoids starting test jobs during *peak hours* (8 am-9:30 am, 12:30 pm-2:30 pm), that is, when most users are likely to start working on Grid'5000 and reserving resources. Second, the scheduler never starts two test jobs simultaneously on the same site.

On the busiest sites, this policy sometimes caused test jobs to be delayed for several days as resources were permanently reserved by users. For some jobs where partial results where particularly useful, specific test configurations for running the test on a single node or on all available nodes were added.

4.4.3 Analyzing and summarizing results

Another need that was not well served by Jenkins alone is the ability to provide a useful summary of the results. There are several different requirements: (1) per test status (for all sites or clusters); (2) per-site or per-cluster status (for all tests); (3) historical perspective.

A per test status is reasonably well provided by Jenkins, but is unfortunately made rather unusable due to the *unstable* status for jobs that were started but could not be scheduled. Jenkins does not provide a way to build a per-site status. Jenkins provides some historical perspective, using weatherlike icons for each job, but it was insufficient for our needs.

To meet those needs, we designed an external status page by exporting data from Jenkins using its REST API[5]. That page (Figure 27) provides a table summarizing the status (current success percentage) for each site and test, and then a list of all failures that can be filtered using basic Javascript. Each failure can be annotated by engineers, for example to mention the corresponding bug.

Using the same method, we wrote a plugin for the Munin monitoring tool to keep historical data about each job and each site (Figure 26).

4.4.4 Why Jenkins, after all?

Due to the large number of Jenkins limitations that were worked-around, one could wonder if using Jenkins as a basis was really a good choice in the first place. We still believe it is, because (1) Jenkins provides a clean execution environment for scripts, with a queue to avoid

[5] Which, it is worth noting, is extremely well designed: it provides a way to get data about all jobs with a configurable level of detail in just one HTTP request.

Co-funded by the Horizon 2020
Framework Programme of the European Union

overloading, the ability for users to trigger manual builds through a web interface; (2) Jenkins provides a storage system for test logs (and optionally, to store artifacts about test runs, such as raw data files, that could be useful in the future), the history of build results, and the ability to browse those test logs through a web interface.

Re-developing those features could have been an option, but would have resulted in significant development work. Additionally, Jenkins is also used for more traditional continuous integration tasks, and it makes sense to keep those CI tasks and testbed testing in the same tool in order to combine them. For example, there are CI tasks that build development versions of software, and then run testbed tests with those development versions installed, in order to evaluate whether those versions are suitable for release.

4.4.5 Test scripts

The goal of test scripts should be to exhibit issues, but also to provide sufficient information to testbed operators to understand and fix the issue. One important limitation of bug reports submitted by users is that issues are described as the users see them, and usually not with all the information that testbed operators would like to have. This problem can be avoided using an automated testing framework only if the scripts are carefully designed in a way that provides that information.

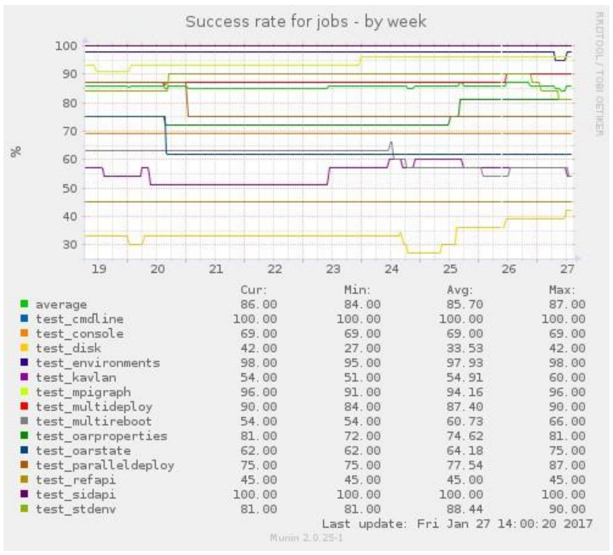
For that reason, we wrote the test scripts using rather simple tools, and performing steps that would be close to what one would use when trying to reproduce a problem manually (which is considered a good practice for automation [13]).

As Grid’5000 is documented through a series of tutorials, an option that was considered was to use the content of those tutorials as the list of actions that would be automated and tested. If successful, it would have ensured that tutorials continue to work over time (which has been a problem in the past). However, this idea was rejected because (1) tutorials are designed with pedagogy in mind, not with test coverage; (2) most tutorials have several options (paths) at one point or another, making it difficult to automate. Therefore, for now we focused on simpler tests that cover the features of the testbed that are known to fail the most frequently, independently from how they are used in tutorials.

At the time of writing, the following tests have been developed and are in production:

- *refapi*: Check (1) the conformance of the description of each node in the Reference API compared to a schema; (2) That nodes of each cluster are homogeneous in terms of hardware configuration, according to their description in the Reference API.
- *oarproperties*: Check that the properties of nodes in the OAR resource manager match what would be generated based on the information in the Reference API.
- *oarstate*: Check the current state of nodes in the OAR resource manager (in particular, check that disabled nodes – due to hardware failure – are correctly documented).
- *cmdline*: Perform basic operations using command-line tools (reservation, deployment). All other tests rely on the Grid’5000 REST API.
- *sidapi*: Perform basic operations using the development branch of the Grid’5000 REST API. All other tests use the stable branch.
- *environments*: Check that each environment maintained by the Grid’5000 team can be provisioned on each cluster, and further check functionality of various features after provisioning (e.g., Internet access from the node).

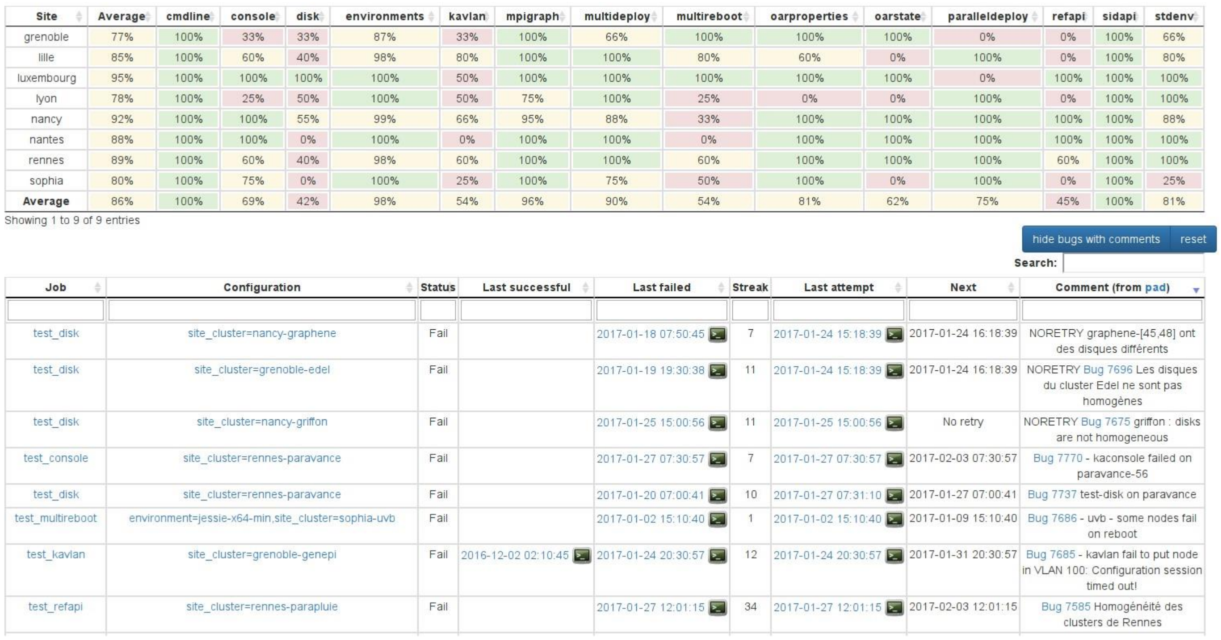
Co-funded by the Horizon 2020 Framework Programme of the European Union



	Cur:	Min:	Avg:	Max:
🟩 average	86.00	84.00	85.70	87.00
🟦 test_cmdline	100.00	100.00	100.00	100.00
🟧 test_console	69.00	69.00	69.00	69.00
🟨 test_disk	42.00	27.00	33.53	42.00
🟦 test_environments	98.00	95.00	97.93	98.00
🟪 test_kavlan	54.00	51.00	54.91	60.00
🟩 test_mpigraph	96.00	91.00	94.16	96.00
🟥 test_multideploy	90.00	84.00	87.40	90.00
⬜ test_multireboot	54.00	54.00	60.73	66.00
🟩 test_oarproperties	81.00	72.00	74.62	81.00
🟦 test_oarstate	62.00	62.00	64.18	75.00
🟫 test_paralleldeploy	75.00	75.00	77.54	87.00
🟨 test_refapi	45.00	45.00	45.00	45.00
🟪 test_sidapi	100.00	100.00	100.00	100.00
🟩 test_stdenv	81.00	81.00	88.44	90.00

Last update: Fri Jan 27 14:00:20 2017

Figure 26: Historical status for each job, as provided by Munin



Site	Average	cmdline	console	disk	environments	kavlan	mpigraph	multideploy	multireboot	oarproperties	oarstate	paralleldeploy	refapi	sidapi	stdenv
grenoble	77%	100%	33%	33%	87%	33%	100%	66%	100%	100%	100%	0%	0%	100%	66%
lille	85%	100%	60%	40%	98%	80%	100%	100%	80%	60%	0%	100%	0%	100%	80%
luxembourg	95%	100%	100%	100%	100%	50%	100%	100%	100%	100%	100%	0%	100%	100%	100%
lyon	78%	100%	25%	50%	100%	50%	75%	100%	25%	0%	0%	100%	0%	100%	100%
nancy	92%	100%	100%	55%	99%	66%	95%	88%	33%	100%	100%	100%	100%	100%	88%
nantes	88%	100%	100%	0%	100%	0%	100%	100%	0%	100%	100%	100%	100%	100%	100%
rennes	89%	100%	60%	40%	98%	60%	100%	100%	60%	100%	100%	100%	60%	100%	100%
sophia	80%	100%	75%	0%	100%	25%	100%	75%	50%	100%	0%	100%	0%	100%	25%
Average	86%	100%	69%	42%	98%	54%	96%	90%	54%	81%	62%	75%	45%	100%	81%

Showing 1 to 9 of 9 entries

Job	Configuration	Status	Last successful	Last failed	Streak	Last attempt	Next	Comment (from pad)
test_disk	site_cluster=nancy-graphene	Fail		2017-01-18 07:50:45	7	2017-01-24 15:18:39	2017-01-24 16:18:39	NORETRY graphene-[45,48] ont des disques différents
test_disk	site_cluster=grenoble-edel	Fail		2017-01-19 19:30:38	11	2017-01-24 15:18:39	2017-01-24 16:18:39	NORETRY Bug 7696 Les disques du cluster Edel ne sont pas homogènes
test_disk	site_cluster=nancy-griffon	Fail		2017-01-25 15:00:56	11	2017-01-25 15:00:56	No retry	NORETRY Bug 7675 griffon : disks are not homogeneous
test_console	site_cluster=rennes-paravance	Fail		2017-01-27 07:30:57	7	2017-01-27 07:30:57	2017-02-03 07:30:57	Bug 7770 - kaconsole failed on paravance-56
test_disk	site_cluster=rennes-paravance	Fail		2017-01-20 07:00:41	10	2017-01-27 07:31:10	2017-01-27 07:00:41	Bug 7737 test-disk on paravance
test_multireboot	environment=jessie-x64-min,site_cluster=sophia-uvb	Fail		2017-01-02 15:10:40	1	2017-01-02 15:10:40	2017-01-09 15:10:40	Bug 7686 - uvb - some nodes fail on reboot
test_kavlan	site_cluster=grenoble-genepi	Fail	2016-12-02 02:10:45	2017-01-24 20:30:57	12	2017-01-24 20:30:57	2017-01-31 20:30:57	Bug 7685 - kavlan fail to put node in VLAN 100: Configuration session timed out!
test_refapi	site_cluster=rennes-parapluie	Fail		2017-01-27 12:01:15	34	2017-01-27 12:01:15	2017-02-03 12:01:15	Bug 7585 Homogénéité des clusters de Rennes

Figure 27: Status page for all tests and sites

- *stdenv*: Similarly to *environments*, perform functional checks in the *standard* environment that is available on nodes without provisioning.

Co-funded by the Horizon 2020
Framework Programme of the European Union

- *paralleldeploy*, *multireboot*, *multideploy*: Perform stress tests on the testbed, ensuring that it is possible to initiate several nodes provisioning operations concurrently, that nodes do not randomly fail to boot
  by rebooting them several times in a row, and that nodes do not randomly fail to deploy by deploying them several times in a row.

- *console*: Check that out-of-band consoles work on every node.
- *kavlan*: Check that network reconfiguration works on every node and network interface.
- *kwapi*: Check that power monitoring works on every node.
- *mpigraph*: Perform a MPI matrix bandwidth test to check connectivity and performance for all nodes of each cluster, and for each network technology (Ethernet, Infiniband, IPoIB).
- *disk*: Check homogeneity of disk configurations (read and write caches) and performance among nodes of the same cluster.
- *dellbios*: Check that BIOS parameters are homogeneous inside a cluster, and that some parameters follow testbed-wide rules (e.g. CPU configuration). Due to public procurement rules in France, most of the Grid'5000 clusters use Dell hardware, which justifies this vendor-specific test.

Overall, this currently results in a total of 751 test configurations. Other tests will be added in the future. The *kwapi* test should be extended to cover network traffic measurements. Other aspects of the testbed's network configuration could be tested, such as MTU settings or support for multicast. Also, at the software level, the tools for automated deployment of OpenStack and Ceph should be tested on all testbeds.

4.5 RESULTS AND DISCUSSION

During the course of this work, 118 bugs were filed in the Grid'5000 bug tracking system, of which 84 have already been fixed. This includes issues that were directly found by the tests, but also problems that were discovered while writing tests, as writing scripts that should run on every cluster and site proved to be a good way to uncover various usability issues (e.g. differences between sites that are not described in the Reference API).

Here are a few examples of issues that were found in the process[6]:

- Several cases of heterogeneous configuration or documentation (different ways to document the same property on different sites) were uncovered by test *refapi* and some other tests (e.g. the *disk* and *dellbios* test): heterogeneous BIOS versions or configuration inside a cluster, disk drives configuration (read or write caching), CPU power configuration settings (C-states), etc. (bugs 7473, 7465, 7675, 7407, 7585, 7370, 7371, 7584, 7586)

[6] The corresponding bugs are available in the Grid'5000 bug tracker (https://intranet.grid5000.fr/bugzilla/). A Grid'5000 account is required, but could be obtained through the Open Access program.

Co-funded by the Horizon 2020
Framework Programme of the European Union

- The *kavlan* test exhibited a number of cabling errors, where two nodes would be inverted in the KaVLAN configuration (bugs 7669, 7580, 7767, 7735, 7381, 7663, 7598, 7515, 7290), and cases where the driver for our network equipment would fail to properly handle error conditions (bug 7637, 7685);
- A number of other weak spots in our infrastructure (unable to handle load, or to work reliably) were uncovered by tests *paralleldeploy*, *multireboot*, *multideploy* (bugs 7482, 7403, 7415, 7686, 7502, 7503). Specifically, the out-of-band consoles service, which was thought to be reliable, was actually failing frequently (bugs 7770, 7362, 7570, 7325, 7466, 7574, 7575, 7411, 7576).
- Some configuration problems were detected in the images that we provide. The network configuration was invalid on one cluster for two images (bug 7342, 7302). The Infiniband stack was randomly failing to start on boot due to an interesting bug (Figure 28).
- A hardware issue causing random reboots on one of the older clusters (the Grenoble adonis cluster). As the warranty had expired, the cluster was shut down.

```
•    local    apps="opensm    osmtest
ibbs ibns" for app in $apps do
•    if ( ps -ef | grep $app | grep -v grep > /dev/null 2>&1 ); then echo "Please stop $app and
all applications running over InfiniBand" echo "Then run \"$0 $ACTION\"" exit 1
•    fi
•    done
```

Figure 28: Excerpt of the openibd script (part of the OFED Infiniband stack, and responsible for starting it during boot). the use of grep $app caused random failures to start, as any unrelated process with e.g. libnss in this command line would cause the test to succeed, and the service to abort start up. The image was updated to a newer version of the OFED script, which switched to using pgrep for more reliable process matching.

Many of the above issues are either issues that are difficult to detect as they do not cause nodes or services to stop functioning, but rather affect experiments in subtle ways; or issues that include an amount of randomness. Very few of those kinds of issues were reported by users.

Two issues in particular are worth explaining in more details.

Boot failures due to a race condition in the kernel (bug 7347): Our work also uncovered bugs that affected more critical pieces of software. While analyzing test failures, we noticed that nodes were taking much longer to boot in about 5% of cases. The problem was tracked down to LVM2 initialization waiting for systemd-udev-settle execution, which is a command in charge of waiting for all physical devices to be initialized that is included as a dependency for LVM2 in the Debian 8 boot sequence. However, due to a race condition in the kernel, a CPU core initialization was hanging, causing systemd-udev-settle to also hang until it reached a timeout. This has been reported to Debian[7]and will be fixed in a future update of the 3.16 branch of the

[7] https://bugs.debian.org/841171

Co-funded by the Horizon 2020
Framework Programme of the European Union

kernel (it was already fixed in Linux 3.19, but Linux 3.16 is the version in Debian 8). This kind of issue shows that the devil is in the detail, and that no piece of software should be considered bug-free.

Heterogeneous disk performance on supposedly identical hardware (bug 7658): The Nantes Grid'5000 site hosts the *econome* cluster, composed of 22 Dell PowerEdge C6220 nodes (a single 2U chassis hosts four servers) with the exact same hardware configuration. On this cluster, the *disk* test showed lower performance on four nodes (from the same chassis): the test workload exhibited an average sequential read bandwidth of 79.3 MB/s on the slow nodes, vs 87.7 MB/s on the fast nodes (a 10% performance difference). Also, the SATA rate advertised by the disk was different (SATA 2.6 on the slow nodes, SATA 3.0 on the fast nodes). It turns out that the slow chassis was bought five months before the rest of the nodes (June 2012 vs November 2012), and, while it was indeed the exact same hardware configuration, it shipped with a different set of BIOS and firmware versions. Due to the lack of tests, this was not detected at the time of the cluster installation. Upgrading the BIOS version on the older nodes did not solve the problem, but upgrading the disk firmwares did.

Obviously, for all experiments performed on this cluster where storage performance had an impact, this puts into question the results that were obtained (as the heterogeneous performance might create results that depend on the placement of data on specific nodes), the repeatability of results (as different nodes from the same cluster would provide different results) and overall, the reproducibility of experiments.

4.6 RELATED WORK

There has been very little work on testbed testing, verification and quality control.

On Grid'5000 itself, it was already mentioned before that each time a node boots, a tool called *g5kchecks* downloads the node's description from the Grid'5000 Reference API, and then verifies using various inventory tools that the visible hardware configuration on the node matches the description from the Reference API [4]. This is complementary to the work described in this section, as *g5kchecks* is not in a position to verify properties at the cluster, site or testbed level (e.g. that clusters are homogeneous). Also, the fact that it runs when the node boots make it timecritical, and thus it cannot run longer performance measurements or tests requiring the collaboration of several nodes. The Emulab-based testbeds (e.g. CloudLab) use a tool that is similar in scope to *g5k-checks* called CheckNode [18].

Also on Emulab, LinkTest [19] can validate the network characteristics of an Emulab experiment (connectivity, latency, bandwidth, link loss, routing).

More generally, this work builds on the products of good practices in software engineering, like software testing [3], continuous integration (CI), continuous delivery (CD) and automation in general. It is similar in some ways to performance regression testing, which is more and more advocated in various communities, e.g. in the Linux kernel development community [20].

Co-funded by the Horizon 2020
Framework Programme of the European Union

4.6.1 REFERENCES

[1] R. D. Peng, "Reproducible research," Coursera lecture, Week 1, Part 2. [Online]. Available: https://www.coursera.org/learn/ reproducible-research

[2] T. Buchert, C. Ruiz, L. Nussbaum, and O. Richard, "A survey of general-purpose experiment management tools for distributed systems," *Future Generation Computer Systems*, vol. 45, pp. 1 – 12, 2015. [Online]. Available: https://hal.inria.fr/hal-01087519

[3] G. J. Myers, C. Sandler, and T. Badgett, *The art of software testing*. John Wiley & Sons, 2011.

[4] D. Margery, E. Morel, L. Nussbaum, O. Richard, and
C. Rohr, "Resources Description, Selection, Reservation and Verification on a Large-scale Testbed," in *TRIDENTCOM -*
9th International Conference on Testbeds and Research Infrastructures for the Development of Networks & Communities, Guangzhou, China, May 2014. [Online].
Available: https://hal.inria.fr/hal-00965708

[5] N. Capit, G. Da Costa, Y. Georgiou, G. Huard, C. Martin, G. Mounie, P. Neyron, and O. Richard, "A batch scheduler´ with high level components," in *Cluster Computing and the Grid, 2005. CCGrid 2005. IEEE International Symposium on*, vol. 2. IEEE, 2005, pp. 776–783.

[6] E. Jeanvoine, L. Sarzyniec, and L. Nussbaum, "Kadeploy3: Efficient and Scalable Operating System Provisioning for Clusters," *USENIX ;login:*, vol. 38, no. 1, pp. 38–44, Feb. 2013. [Online]. Available: https://hal.inria.fr/hal-00909111

[7] D. Balouek, A. Carpen Amarie, G. Charrier, F. Desprez,
E. Jeannot, E. Jeanvoine, A. Lebre, D. Margery, N. Niclausse,` L. Nussbaum, O. Richard, C. Perez, F. Quesnel, C. Rohr,´ and L. Sarzyniec, "Adding virtualization capabilities to the Grid'5000 testbed," in *Cloud Computing and Services Science*, ser. Communications in Computer and Information Science, I. Ivanov, M. Sinderen, F. Leymann, and T. Shan, Eds. Springer International Publishing, 2013, vol. 367, pp. 3–20.

[8] F. Clouet, S. Delamare, J.-P. Gelas, L. Lefevre, L. Nussbaum,` C. Parisot, L. Pouilloux, and F. Rossigneux, "A Unified
Monitoring Framework for Energy Consumption and Network Traffic," in *TRIDENTCOM - International Conference on Testbeds and Research Infrastructures for the Development of Networks & Communities*, Vancouver, Canada, Jun. 2015, p. 10. [Online]. Available: https://hal.inria.fr/hal-01167915

[9] S. Tatham, "How to report bugs effectively,"
1999. [Online]. Available: http://www.chiark.greenend.org. uk/~sgtatham/bugs.html

[10] E. S. Raymond and R. Moen, "How to ask questions the smart way." [Online]. Available: http://www.catb.org/esr/faqs/ smart-questions.html

[11] B. Gregg, "Visualizing system latency," *Commun. ACM*, vol. 53, no. 7, pp. 48–54, Jul. 2010. [Online]. Available:
http://doi.acm.org/10.1145/1785414.1785435

[12] "Jenkins – build great things at scale." [Online]. Available: https://jenkins.io/

[13] T. A. Limoncelli, "Automation should be like iron man, not ultron," *Queue*, vol. 13, no. 8, pp. 50:50–50:59, Sep. 2015. [Online]. Available: http://doi.acm.org/10.1145/ 2838344.2841313

[14] "Fed4fire – monitoring." [Online]. Available: https: //flsmonitor.fed4fire.eu/

[15] B. Vermeulen, "Fed4fire d2.8 – third integration and testing roadmap," 2016. [Online]. Available: https://www.fed4fire.eu/wp-content/uploads/2016/10/ d2-8-third-integration-and-testing-roadmap.pdf

[16] "Fed4fire federation monitor." [Online]. Available: https: //fedmon.fed4fire.eu/

[17] "Geni am api acceptance tests." [Online]. Available: http://trac.gpolab.bbn.com/gcf/wiki/AmApiAcceptanceTests

[18] "Emulab – checknode." [Online]. Available: https://wiki. emulab.net/wiki/checknode

[19] "Emulab – linktest." [Online]. Available: https://wiki.emulab. net/wiki/linktest

[20] D. Bueso, "Performance monitoring in the linux kernel," in *Linux Plumbers Conference*, 2015. [Online]. Available: http://events.linuxfoundation.org/sites/events/files/ slides/dbueso-lpc-2015-kperfmonitor.pdf

Co-funded by the Horizon 2020
Framework Programme of the European Union

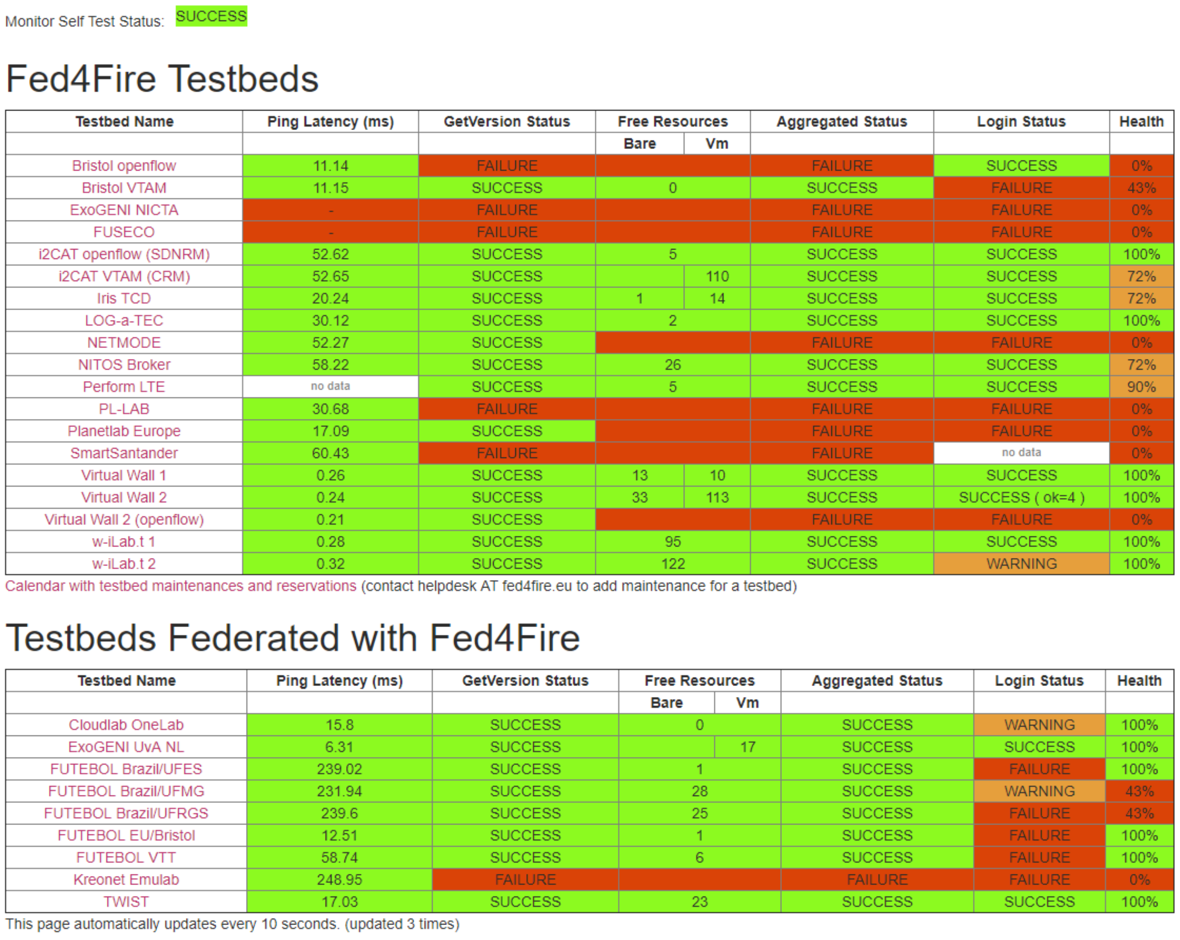
5 FEDERATION MONITORING

5.1 INTRODUCTION

Because the federation of testbed is a ‘loose’ federation (there is no single entity controlling all testbeds), continuous monitoring of the testbeds is key. In that way, both the testbed operators can be warned if something goes wrong and the experimenters can have an overview of which testbeds are okay to use.

5.2 FED4FIRE FLSMONITORING

Already in Fed4FIRE we had a first version of federation monitoring, a screenshot of the front-end dashboard can be seen in Figure 29 and can be found live at https://flsmonitor.fed4fire.eu/fls (it was then known as First Level Support (FLS) monitoring).



Monitor Self Test Status: SUCCESS

Fed4Fire Testbeds

Testbed Name	Ping Latency (ms)	GetVersion Status	Free Resources		Aggregated Status	Login Status	Health
			Bare	Vm			
Bristol openflow	11.14	FAILURE			FAILURE	SUCCESS	0%
Bristol VTAM	11.15	SUCCESS	0		SUCCESS	FAILURE	43%
ExoGENI NICTA	-	FAILURE			FAILURE	FAILURE	0%
FUSECO	-	FAILURE			FAILURE	FAILURE	0%
i2CAT openflow (SDNRM)	52.62	SUCCESS	5		SUCCESS	SUCCESS	100%
i2CAT VTAM (CRM)	52.65	SUCCESS		110	SUCCESS	SUCCESS	72%
Iris TCD	20.24	SUCCESS	1	14	SUCCESS	SUCCESS	72%
LOG-a-TEC	30.12	SUCCESS	2		SUCCESS	SUCCESS	100%
NETMODE	52.27	SUCCESS			FAILURE	FAILURE	0%
NITOS Broker	58.22	SUCCESS	26		SUCCESS	SUCCESS	72%
Perform LTE	no data	SUCCESS	5		SUCCESS	SUCCESS	90%
PL-LAB	30.68	FAILURE			FAILURE	FAILURE	0%
Planetlab Europe	17.09	SUCCESS			FAILURE	FAILURE	0%
SmartSantander	60.43	FAILURE			FAILURE	no data	0%
Virtual Wall 1	0.26	SUCCESS	13	10	SUCCESS	SUCCESS	100%
Virtual Wall 2	0.24	SUCCESS	33	113	SUCCESS	SUCCESS (ok=4)	100%
Virtual Wall 2 (openflow)	0.21	SUCCESS			FAILURE	FAILURE	0%
w-iLab.t 1	0.28	SUCCESS	95		SUCCESS	SUCCESS	100%
w-iLab.t 2	0.32	SUCCESS	122		SUCCESS	WARNING	100%

Calendar with testbed maintenances and reservations (contact helpdesk AT fed4fire.eu to add maintenance for a testbed)

Testbeds Federated with Fed4Fire

Testbed Name	Ping Latency (ms)	GetVersion Status	Free Resources		Aggregated Status	Login Status	Health
			Bare	Vm			
Cloudlab OneLab	15.8	SUCCESS	0		SUCCESS	WARNING	100%
ExoGENI UvA NL	6.31	SUCCESS		17	SUCCESS	SUCCESS	100%
FUTEBOL Brazil/UFES	239.02	SUCCESS	1		SUCCESS	FAILURE	100%
FUTEBOL Brazil/UFMG	231.94	SUCCESS	28		SUCCESS	WARNING	43%
FUTEBOL Brazil/UFRGS	239.6	SUCCESS	25		SUCCESS	FAILURE	43%
FUTEBOL EU/Bristol	12.51	SUCCESS	1		SUCCESS	FAILURE	100%
FUTEBOL VTT	58.74	SUCCESS	6		SUCCESS	FAILURE	100%
Kreonet Emulab	248.95	FAILURE			FAILURE	FAILURE	0%
TWIST	17.03	SUCCESS	23		SUCCESS	SUCCESS	100%

This page automatically updates every 10 seconds. (updated 3 times)

Figure 29: Screenshot of federation monitoring in the Fed4FIRE project (2012-2016)

Co-funded by the Horizon 2020
Framework Programme of the European Union

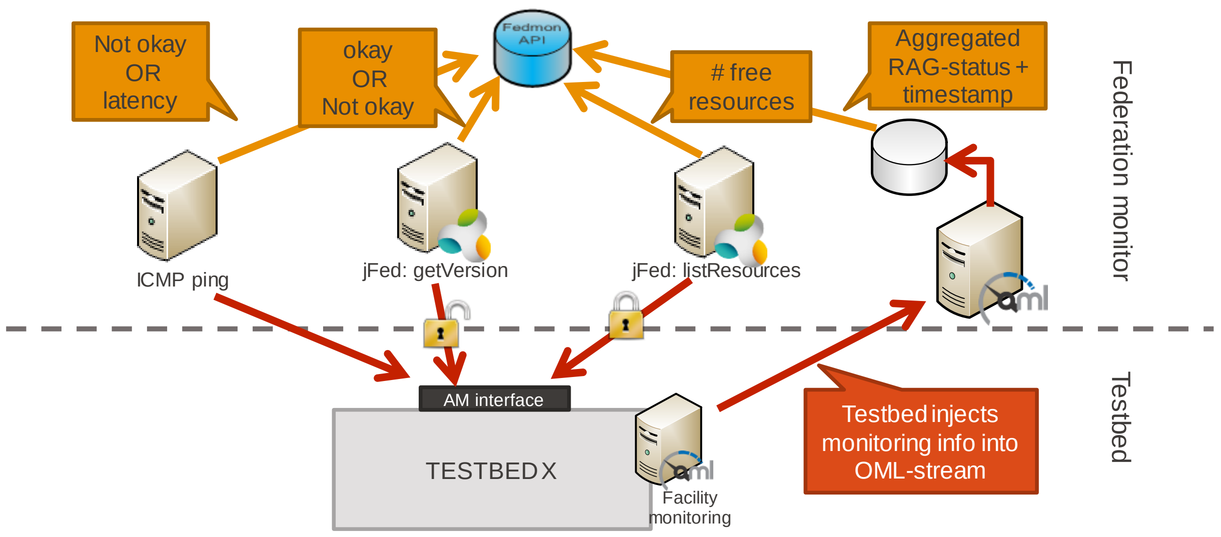


Figure 30: Basic monitoring setup

Figure 30 shows the basic inputs for that monitoring:

- We ping the AM (Aggregate Manager) API endpoint of the testbed to know if there is a general networking problem or not
- After that, we do a GetVersion call which is an unauthenticated call which learns us if the API is up and running or not
- The listRresources call tests if the authentication is okay and gives us also the number of free resources
- In Fed4FIRE we also had internal facility monitoring (e.g. Zabbix, Nagios, …) of which the testbed sent information to the monitoring. This has been removed mostly from the monitoring as not that many testbeds sent that information (especially from other federations).

All these tests run each 15-60 minutes and result in a Red-Amber-Green status called the ‘Aggregated Status’.

Apart from these API calls, there is also run once or twice a day a full experiment with a single node, including ssh login. This the login status. From the history of the aggregated status and login status, we create the health status. E.g. if a login test fails once in the last 31 days, the health is 90%. If it fails twice in the last 7 days, it is 72%, etc.

5.3 FED4FIRE+ NEW MONITORING

In Fed4FIRE+ we overhauled the federation monitoring for the following reasons:

- To make the back-end more scalable. There are more and more tests (see further) and more and more testbeds. Some tests take more time or timeout, so more tests in parallel are ongoing. This scheduling complexity cannot be handled by existing

Co-funded by the Horizon 2020 Framework Programme of the European Union

frameworks as Jenkins, so we have improved our basic scheduler of Fed4FIRE to cope with this.

- To make the front-end more appealing and scalable for the amount of testbeds being monitored
- To have an API to the monitoring, so user tools as jFed can use the information to present to the end users. Other tools can also get the information as needed.
- To add more specific tests (e.g. test each node of a testbed), see further.

The new front-end can be found at https://fedmon.fed4fire.eu and in Figure 31. The selectors on the left can be used to only show particular types of testbeds. The new monitor has also a map view (Figure 32).

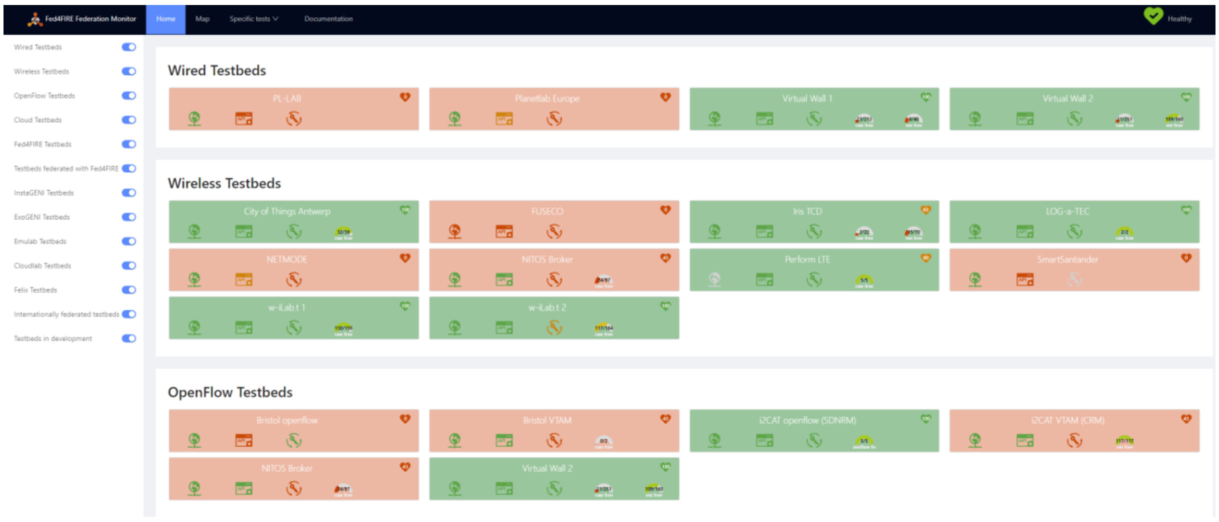
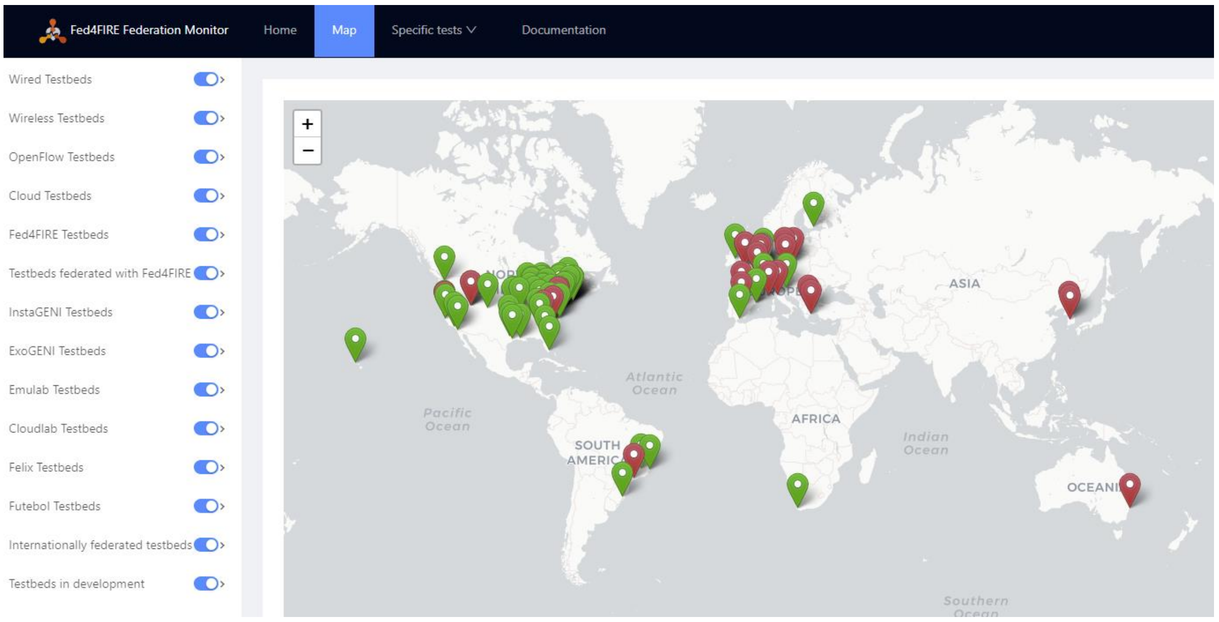


Figure 31: Screenshot of front-end of https://fedmon.fed4fire.eu



Co-funded by the Horizon 2020 Framework Programme of the European Union

Figure 32: Map view of https://fedmon.fed4fire.eu

Figure 33 and Figure 34 show more details on the monitoring icons available per testbed, while **Error! Reference source not found.** shows a zoomed out screenshot of almost all testbeds monitored.

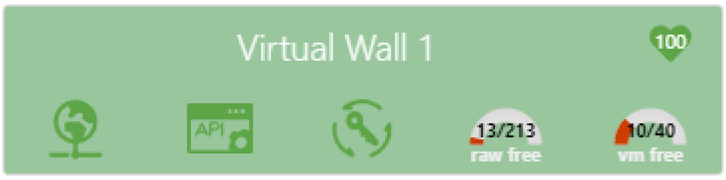
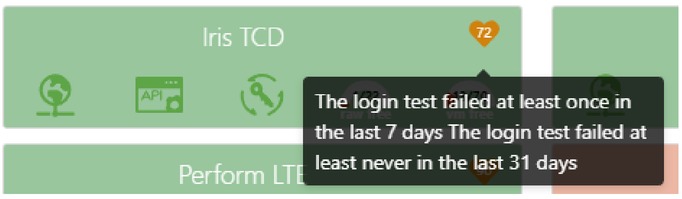


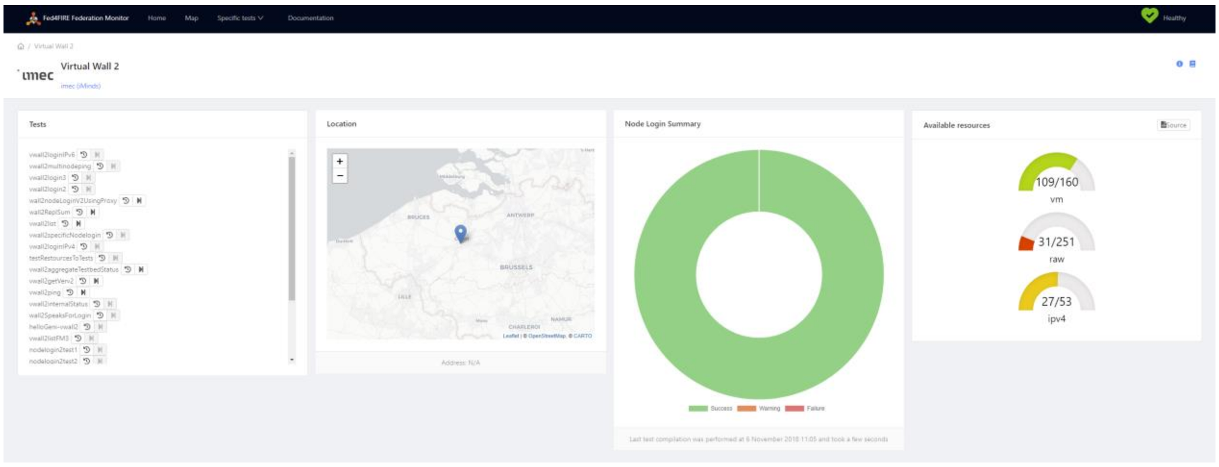
Figure 33: Icon overview of a single testbed (ping, getversion/listresources, login test, number of free and total resources, health status)



The login test failed at least once in the last 7 days The login test failed at least never in the last 31 days

Figure 34: When hovering over the icons, the user can get more information, e.g. why the health status is not 100%

When clicking on a testbed, one comes into a detailed overview per testbed of the tests being run (Figure 35), an overview of the resources (and evolution over time, see Figure 36), and an overview of the testbed availability during the last year (Figure 37).



vwall2loginIPv6
vwall2multinodeping
vwall2login3
vwall2login2
wall2nodeLoginV2UsingProxy
wall2ReplSum
vwall2list
vwall2specificNodelogin
vwall2loginIPv4
testRestourcesToTests
vwall2aggregateTestbedStatus
vwall2getVerv2
vwall2ping
vwall2internalStatus
wall2SpeaksForLogin
helloGeni-vwall2
vwall2listFM3
nodelogin2test1
nodelogin2test2

Address: N/A

Last test compilation was performed at 6 November 2018 11:05 and took a few seconds

Figure 35: More detailed information per testbed

Co-funded by the Horizon 2020 Framework Programme of the European Union

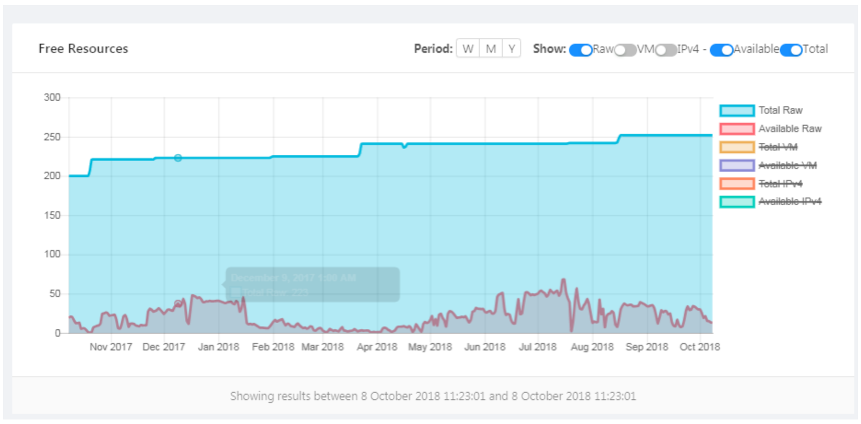


Figure 36: Evolution over time of free and total resources of a testbed

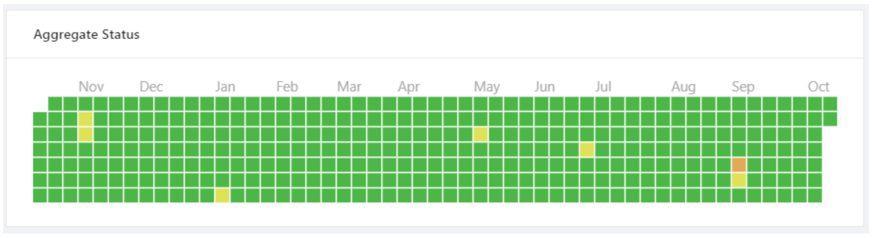


Figure 37: Evolution over time of the testbed uptime (1 square is a single day)

Co-funded by the Horizon 2020
Framework Programme of the European Union

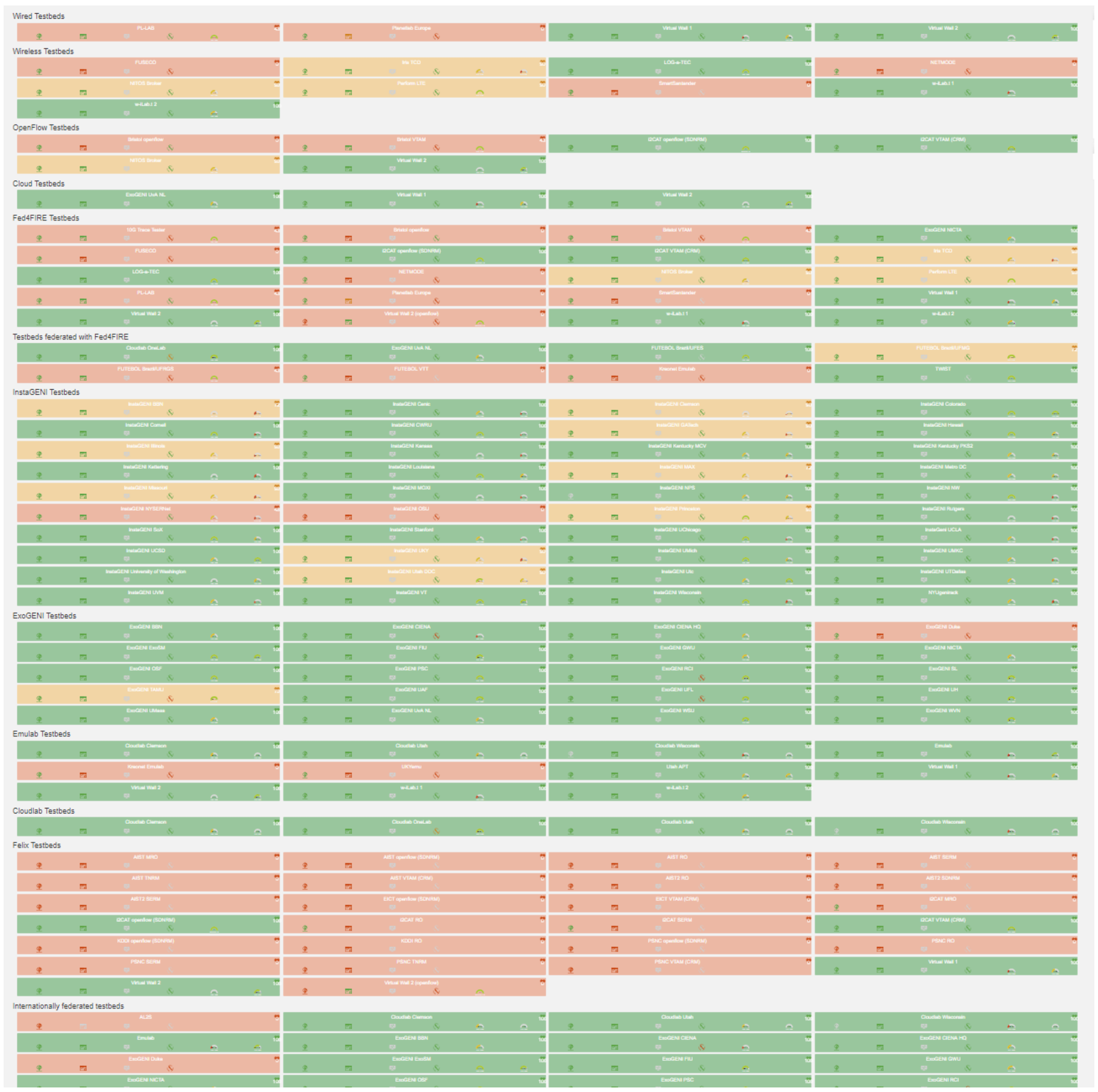
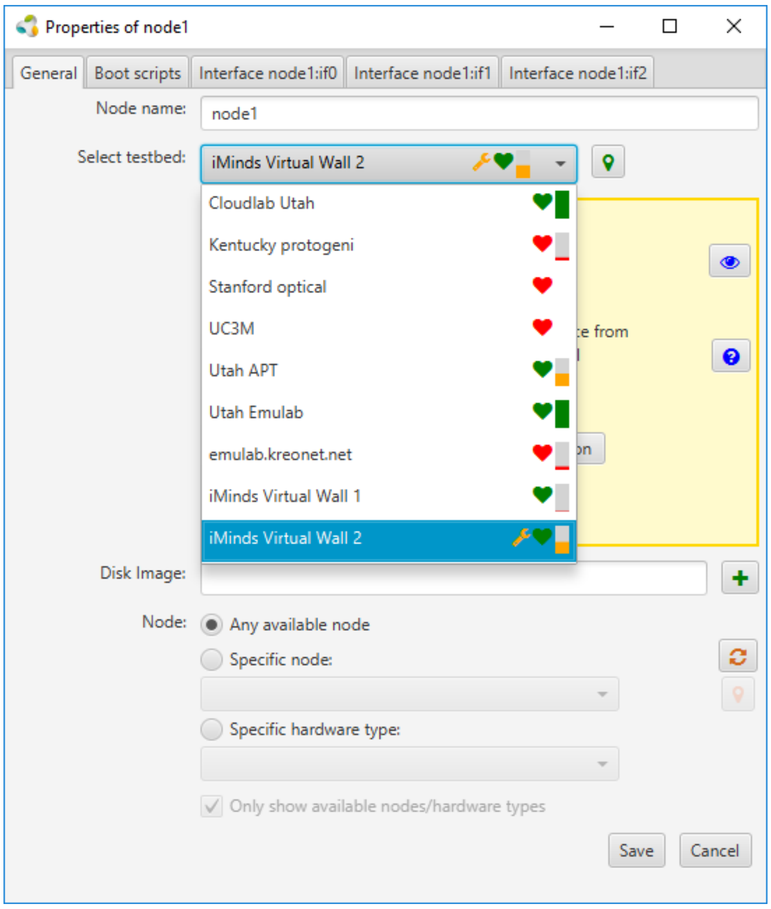


Figure 38: Zoomed out overview of almost all monitored testbeds

5.4 FEEDBACK OF THE MONITORING TO THE USER

As also described in D2.2, we want to show all this monitoring information also in an easy way to the users. For this, an API was created on top of the monitoring framework, and through that API, the information about the testbeds is fetched by jFed and shown to the users. See Figure 39 and Figure 40.

Co-funded by the Horizon 2020 Framework Programme of the European Union



Node name: node1

Select testbed: iMinds Virtual Wall 2

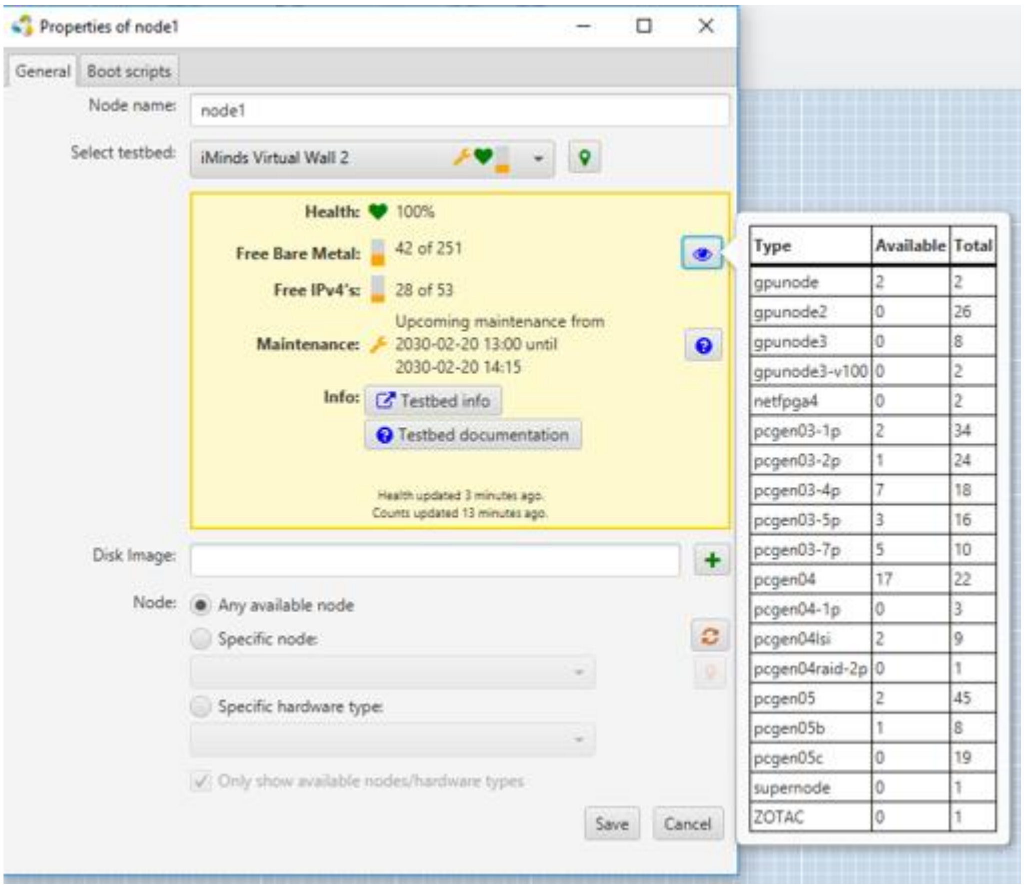
- Cloudlab Utah
- Kentucky protogeni
- Stanford optical
- UC3M
- Utah APT
- Utah Emulab
- emulab.kreonet.net
- iMinds Virtual Wall 1
- iMinds Virtual Wall 2

Node:
- [✓ Any available node]
- Specific node:
- Specific hardware type:

✓ Only show available nodes/hardware types

Figure 39: Testbed and resource availability visible in jFed

Co-funded by the Horizon 2020
Framework Programme of the European Union



Node name: node1

Select testbed: iMinds Virtual Wall 2

Health: 100%

Free Bare Metal: 42 of 251

Free IPv4's: 28 of 53

Maintenance: Upcoming maintenance from 2030-02-20 13:00 until 2030-02-20 14:15

Info:

Health updated 3 minutes ago.
Counts updated 13 minutes ago.

Disk Image:

Node: [✓ Any available node] Specific node: Specific hardware type:

✓ Only show available nodes/hardware types

Type	Available	Total
gpunode	2	2
gpunode2	0	26
gpunode3	0	8
gpunode3-v100	0	2
netfpga4	0	2
pcgen03-1p	2	34
pcgen03-2p	1	24
pcgen03-4p	7	18
pcgen03-5p	3	16
pcgen03-7p	5	10
pcgen04	17	22
pcgen04-1p	0	3
pcgen04lsi	2	9
pcgen04raid-2p	0	1
pcgen05	2	45
pcgen05b	1	8
pcgen05c	0	19
supernode	0	1
ZOTAC	0	1

Figure 40: jFed availability resources and resources per hardware type

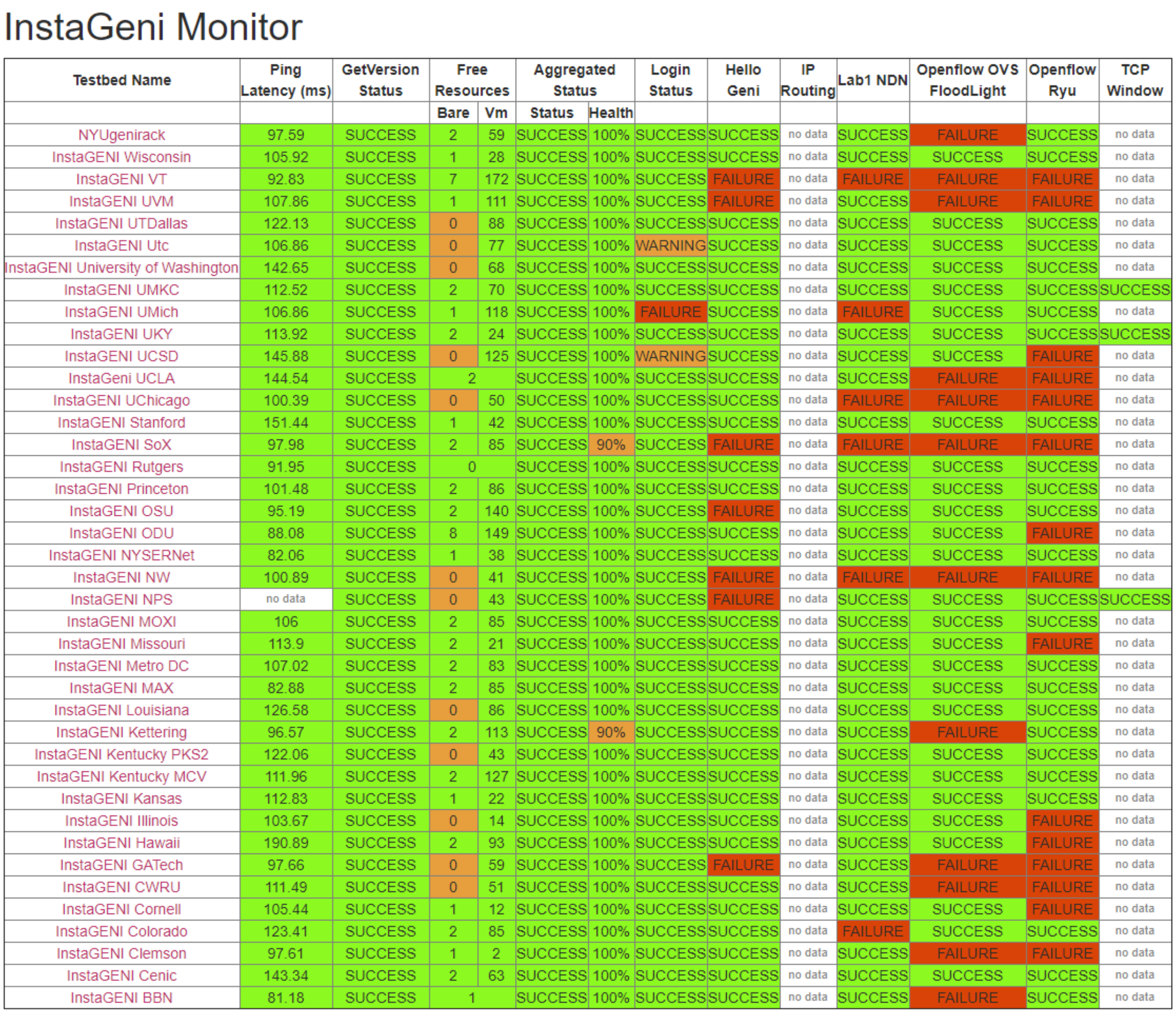
5.5 SPECIFIC TESTS

In Fed4FIRE+ we added also specific and more advanced tests (although the node ssh login test is already advanced, it does not test deploying software on the node or using multi-node setups). These specific tests can be reached from the top menu bar at https://fedmon.fed4fire.eu.

5.5.1 GENI tutorial testing

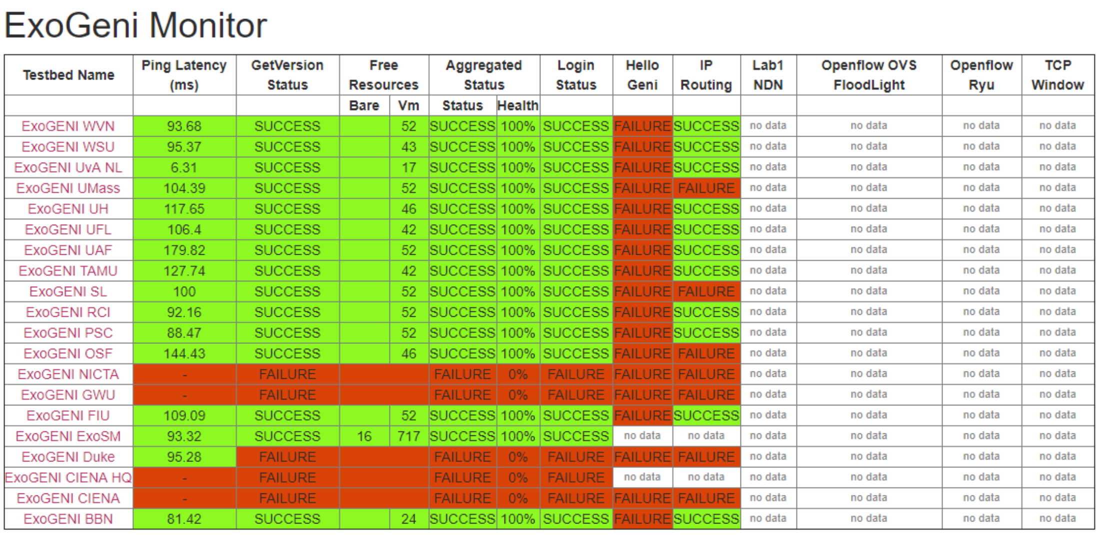
There was a demand in both GENI and Fed4FIRE to be able to test the tutorials typically offered to new users. Especially in GENI sometimes a tutorial failed because a specific testbed (GENI rack) did miss a needed image or configuration. The overview of GENI tutorial testing can be found at: https://flsmonitor.fed4fire.eu/genitests and in Figure 41. Ansible is used to automate these tests.

Co-funded by the Horizon 2020
Framework Programme of the European Union



InstaGeni Monitor

Testbed Name	Ping Latency (ms)	GetVersion Status	Free Resources		Aggregated Status		Login Status	Hello Geni	IP Routing	Lab1 NDN	Openflow OVS FloodLight	Openflow Ryu	TCP Window
			Bare	Vm	Status	Health							
NYUgenirack	97.59	SUCCESS	2	59	SUCCESS	100%	SUCCESS	SUCCESS	no data	SUCCESS	FAILURE	SUCCESS	no data
InstaGENI Wisconsin	105.92	SUCCESS	1	28	SUCCESS	100%	SUCCESS	SUCCESS	no data	SUCCESS	SUCCESS	SUCCESS	no data
InstaGENI VT	92.83	SUCCESS	7	172	SUCCESS	100%	SUCCESS	FAILURE	no data	FAILURE	FAILURE	FAILURE	no data
InstaGENI UVM	107.86	SUCCESS	1	111	SUCCESS	100%	SUCCESS	FAILURE	no data	SUCCESS	FAILURE	FAILURE	no data
InstaGENI UTDallas	122.13	SUCCESS	0	88	SUCCESS	100%	SUCCESS	SUCCESS	no data	SUCCESS	SUCCESS	SUCCESS	no data
InstaGENI Utc	106.86	SUCCESS	0	77	SUCCESS	100%	WARNING	SUCCESS	no data	SUCCESS	SUCCESS	SUCCESS	no data
InstaGENI University of Washington	142.65	SUCCESS	0	68	SUCCESS	100%	SUCCESS	SUCCESS	no data	SUCCESS	SUCCESS	SUCCESS	no data
InstaGENI UMKC	112.52	SUCCESS	2	70	SUCCESS	100%	SUCCESS	SUCCESS	no data	SUCCESS	SUCCESS	SUCCESS	SUCCESS
InstaGENI UMich	106.86	SUCCESS	1	118	SUCCESS	100%	FAILURE	SUCCESS	no data	FAILURE	SUCCESS	SUCCESS	no data
InstaGENI UKY	113.92	SUCCESS	2	24	SUCCESS	100%	SUCCESS	SUCCESS	no data	SUCCESS	SUCCESS	SUCCESS	SUCCESS
InstaGENI UCSD	145.88	SUCCESS	0	125	SUCCESS	100%	WARNING	SUCCESS	no data	SUCCESS	SUCCESS	FAILURE	no data
InstaGeni UCLA	144.54	SUCCESS	2		SUCCESS	100%	SUCCESS	SUCCESS	no data	SUCCESS	FAILURE	FAILURE	no data
InstaGENI UChicago	100.39	SUCCESS	0	50	SUCCESS	100%	SUCCESS	SUCCESS	no data	FAILURE	FAILURE	FAILURE	no data
InstaGENI Stanford	151.44	SUCCESS	1	42	SUCCESS	100%	SUCCESS	SUCCESS	no data	SUCCESS	SUCCESS	SUCCESS	no data
InstaGENI SoX	97.98	SUCCESS	2	85	SUCCESS	90%	SUCCESS	FAILURE	no data	FAILURE	FAILURE	FAILURE	no data
InstaGENI Rutgers	91.95	SUCCESS	0		SUCCESS	100%	SUCCESS	SUCCESS	no data	SUCCESS	SUCCESS	SUCCESS	no data
InstaGENI Princeton	101.48	SUCCESS	2	86	SUCCESS	100%	SUCCESS	SUCCESS	no data	SUCCESS	SUCCESS	SUCCESS	no data
InstaGENI OSU	95.19	SUCCESS	2	140	SUCCESS	100%	SUCCESS	FAILURE	no data	SUCCESS	SUCCESS	SUCCESS	no data
InstaGENI ODU	88.08	SUCCESS	8	149	SUCCESS	100%	SUCCESS	SUCCESS	no data	SUCCESS	SUCCESS	FAILURE	no data
InstaGENI NYSERNet	82.06	SUCCESS	1	38	SUCCESS	100%	SUCCESS	SUCCESS	no data	SUCCESS	SUCCESS	SUCCESS	no data
InstaGENI NW	100.89	SUCCESS	0	41	SUCCESS	100%	SUCCESS	FAILURE	no data	FAILURE	FAILURE	FAILURE	no data
InstaGENI NPS	no data	SUCCESS	0	43	SUCCESS	100%	SUCCESS	FAILURE	no data	SUCCESS	SUCCESS	SUCCESS	SUCCESS
InstaGENI MOXI	106	SUCCESS	2	85	SUCCESS	100%	SUCCESS	SUCCESS	no data	SUCCESS	SUCCESS	SUCCESS	no data
InstaGENI Missouri	113.9	SUCCESS	2	21	SUCCESS	100%	SUCCESS	SUCCESS	no data	SUCCESS	SUCCESS	FAILURE	no data
InstaGENI Metro DC	107.02	SUCCESS	2	83	SUCCESS	100%	SUCCESS	SUCCESS	no data	SUCCESS	SUCCESS	SUCCESS	no data
InstaGENI MAX	82.88	SUCCESS	2	85	SUCCESS	100%	SUCCESS	SUCCESS	no data	SUCCESS	SUCCESS	SUCCESS	no data
InstaGENI Louisiana	126.58	SUCCESS	0	86	SUCCESS	100%	SUCCESS	SUCCESS	no data	SUCCESS	SUCCESS	SUCCESS	no data
InstaGENI Kettering	96.57	SUCCESS	2	113	SUCCESS	90%	SUCCESS	SUCCESS	no data	SUCCESS	FAILURE	SUCCESS	no data
InstaGENI Kentucky PKS2	122.06	SUCCESS	0	43	SUCCESS	100%	SUCCESS	SUCCESS	no data	SUCCESS	SUCCESS	SUCCESS	no data
InstaGENI Kentucky MCV	111.96	SUCCESS	2	127	SUCCESS	100%	SUCCESS	SUCCESS	no data	SUCCESS	SUCCESS	SUCCESS	no data
InstaGENI Kansas	112.83	SUCCESS	1	22	SUCCESS	100%	SUCCESS	SUCCESS	no data	SUCCESS	SUCCESS	SUCCESS	no data
InstaGENI Illinois	103.67	SUCCESS	0	14	SUCCESS	100%	SUCCESS	SUCCESS	no data	SUCCESS	SUCCESS	FAILURE	no data
InstaGENI Hawaii	190.89	SUCCESS	2	93	SUCCESS	100%	SUCCESS	SUCCESS	no data	SUCCESS	SUCCESS	FAILURE	no data
InstaGENI GATech	97.66	SUCCESS	0	59	SUCCESS	100%	SUCCESS	FAILURE	no data	SUCCESS	FAILURE	FAILURE	no data
InstaGENI CWRU	111.49	SUCCESS	0	51	SUCCESS	100%	SUCCESS	SUCCESS	no data	SUCCESS	FAILURE	FAILURE	no data
InstaGENI Cornell	105.44	SUCCESS	1	12	SUCCESS	100%	SUCCESS	SUCCESS	no data	SUCCESS	SUCCESS	FAILURE	no data
InstaGENI Colorado	123.41	SUCCESS	2	85	SUCCESS	100%	SUCCESS	SUCCESS	no data	FAILURE	SUCCESS	SUCCESS	no data
InstaGENI Clemson	97.61	SUCCESS	1	2	SUCCESS	100%	SUCCESS	SUCCESS	no data	SUCCESS	FAILURE	FAILURE	no data
InstaGENI Cenic	143.34	SUCCESS	2	63	SUCCESS	100%	SUCCESS	SUCCESS	no data	SUCCESS	SUCCESS	SUCCESS	no data
InstaGENI BBN	81.18	SUCCESS	1		SUCCESS	100%	SUCCESS	SUCCESS	no data	SUCCESS	FAILURE	SUCCESS	no data

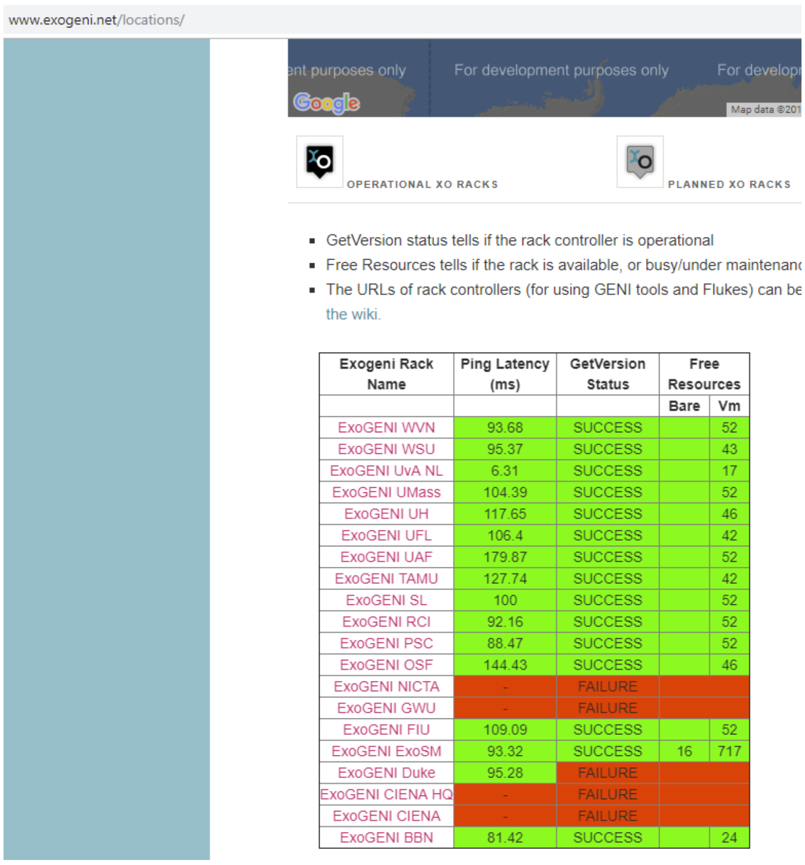


ExoGeni Monitor

Testbed Name	Ping Latency (ms)	GetVersion Status	Free Resources		Aggregated Status		Login Status	Hello Geni	IP Routing	Lab1 NDN	Openflow OVS FloodLight	Openflow Ryu	TCP Window
			Bare	Vm	Status	Health							
ExoGENI WVN	93.68	SUCCESS		52	SUCCESS	100%	SUCCESS	FAILURE	SUCCESS	no data	no data	no data	no data
ExoGENI WSU	95.37	SUCCESS		43	SUCCESS	100%	SUCCESS	FAILURE	SUCCESS	no data	no data	no data	no data
ExoGENI UvA NL	6.31	SUCCESS		17	SUCCESS	100%	SUCCESS	FAILURE	SUCCESS	no data	no data	no data	no data
ExoGENI UMass	104.39	SUCCESS		52	SUCCESS	100%	SUCCESS	FAILURE	FAILURE	no data	no data	no data	no data
ExoGENI UH	117.65	SUCCESS		46	SUCCESS	100%	SUCCESS	FAILURE	SUCCESS	no data	no data	no data	no data
ExoGENI UFL	106.4	SUCCESS		42	SUCCESS	100%	SUCCESS	FAILURE	SUCCESS	no data	no data	no data	no data
ExoGENI UAF	179.82	SUCCESS		52	SUCCESS	100%	SUCCESS	FAILURE	SUCCESS	no data	no data	no data	no data
ExoGENI TAMU	127.74	SUCCESS		42	SUCCESS	100%	SUCCESS	FAILURE	SUCCESS	no data	no data	no data	no data
ExoGENI SL	100	SUCCESS		52	SUCCESS	100%	SUCCESS	FAILURE	FAILURE	no data	no data	no data	no data
ExoGENI RCI	92.16	SUCCESS		52	SUCCESS	100%	SUCCESS	FAILURE	SUCCESS	no data	no data	no data	no data
ExoGENI PSC	88.47	SUCCESS		52	SUCCESS	100%	SUCCESS	FAILURE	SUCCESS	no data	no data	no data	no data
ExoGENI OSF	144.43	SUCCESS		46	SUCCESS	100%	SUCCESS	FAILURE	FAILURE	no data	no data	no data	no data
ExoGENI NICTA	-	FAILURE			FAILURE	0%	FAILURE	FAILURE	FAILURE	no data	no data	no data	no data
ExoGENI GWU	-	FAILURE			FAILURE	0%	FAILURE	FAILURE	FAILURE	no data	no data	no data	no data
ExoGENI FIU	109.09	SUCCESS		52	SUCCESS	100%	SUCCESS	FAILURE	SUCCESS	no data	no data	no data	no data
ExoGENI ExoSM	93.32	SUCCESS	16	717	SUCCESS	100%	SUCCESS	no data	no data	no data	no data	no data	no data
ExoGENI Duke	95.28	FAILURE			FAILURE	0%	FAILURE	FAILURE	FAILURE	no data	no data	no data	no data
ExoGENI CIENA HQ	-	FAILURE			FAILURE	0%	FAILURE	no data	no data	no data	no data	no data	no data
ExoGENI CIENA	-	FAILURE			FAILURE	0%	FAILURE	FAILURE	FAILURE	no data	no data	no data	no data
ExoGENI BBN	81.42	SUCCESS		24	SUCCESS	100%	SUCCESS	FAILURE	SUCCESS	no data	no data	no data	no data

Co-funded by the Horizon 2020
Framework Programme of the European Union

Figure 41: GENI tutorial testing


www.exogeni.net/locations/

- GetVersion status tells if the rack controller is operational
- Free Resources tells if the rack is available, or busy/under maintenanc
- The URLs of rack controllers (for using GENI tools and Flukes) can be the wiki.

Exogeni Rack Name	Ping Latency (ms)	GetVersion Status	Free Resources	
			Bare	Vm
ExoGENI WVN	93.68	SUCCESS		52
ExoGENI WSU	95.37	SUCCESS		43
ExoGENI UvA NL	6.31	SUCCESS		17
ExoGENI UMass	104.39	SUCCESS		52
ExoGENI UH	117.65	SUCCESS		46
ExoGENI UFL	106.4	SUCCESS		42
ExoGENI UAF	179.87	SUCCESS		52
ExoGENI TAMU	127.74	SUCCESS		42
ExoGENI SL	100	SUCCESS		52
ExoGENI RCI	92.16	SUCCESS		52
ExoGENI PSC	88.47	SUCCESS		52
ExoGENI OSF	144.43	SUCCESS		46
ExoGENI NICTA	-	FAILURE		
ExoGENI GWU	-	FAILURE		
ExoGENI FIU	109.09	SUCCESS		52
ExoGENI ExoSM	93.32	SUCCESS	16	717
ExoGENI Duke	95.28	FAILURE		
ExoGENI CIENA HQ	-	FAILURE		
ExoGENI CIENA	-	FAILURE		
ExoGENI BBN	81.42	SUCCESS		24

Figure 42: Exogeni monitoring integrated in the www.exogeni.net webpage

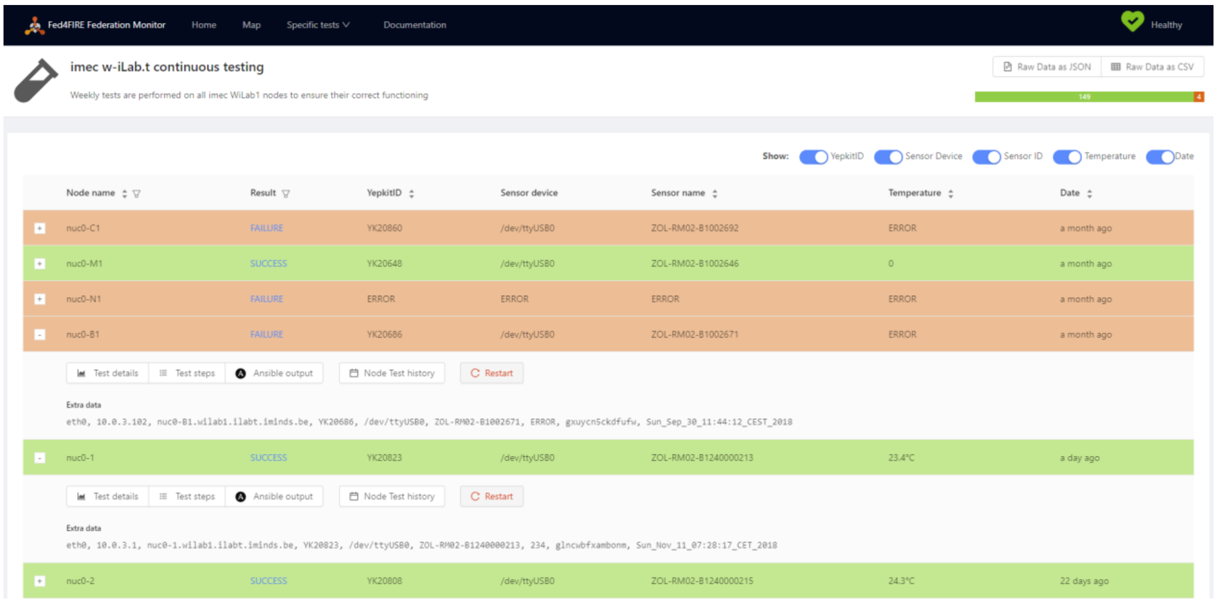
5.5.2 Testing specific devices, e.g. IoT sensors

For certain testbeds (e.g. wireless testbeds, planetlab), users chose specific nodes to use. So it is important that we know for each node if it works or not. For w-iLab.t 1, there is also a Zolertia re-mote IoT device attached to each linux node. We wanted to verify if the linux nodes and Zolertia devices are operational or not. At https://fedmon.fed4fire.eu/wilab/ one can find an overview of these results. Each week, it is tried to verify all devices.

The output in Figure 43 shows per line the result of a single node:

- Green means all is fine, serial numbers are printed, a temperature reading is shown to verify that the firmware flashing to the IoT sensor has been correct and the date when this node was last tested is shown. (if a node is in use, then it can't be tested e.g.).
- Red means something is wrong.
- Right top shows also an overview, e.g. 149 devices are working correctly, 4 are broken.
- When clicking the + at the left (Figure 44), more details can be seen (e.g. detailed test output, test history to see when it was broken, etc)

Co-funded by the Horizon 2020
Framework Programme of the European Union



imec w-iLab.t continuous testing

Weekly tests are performed on all imec WiLab1 nodes to ensure their correct functioning

	Node name	Result	YepkitID	Sensor device	Sensor name	Temperature	Date
+	nuc0-C1	FAILURE	YK20860	/dev/ttyUSB0	ZOL-RM02-B1002692	ERROR	a month ago
+	nuc0-M1	SUCCESS	YK20648	/dev/ttyUSB0	ZOL-RM02-B1002646	0	a month ago
+	nuc0-N1	FAILURE	ERROR	ERROR	ERROR	ERROR	a month ago
-	nuc0-B1	FAILURE	YK20686	/dev/ttyUSB0	ZOL-RM02-B1002671	ERROR	a month ago

Extra data

```
eth0, 10.0.3.102, nuc0-B1.wilab1.ilabt.iminds.be, YK20686, /dev/ttyUSB0, ZOL-RM02-B1002671, ERROR, gxuycn5ckdfufw, Sun_Sep_30_11:44:12_CEST_2018
```

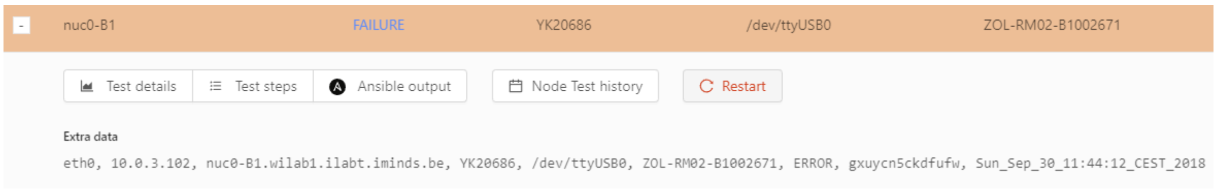
	Node name	Result	YepkitID	Sensor device	Sensor name	Temperature	Date
-	nuc0-1	SUCCESS	YK20823	/dev/ttyUSB0	ZOL-RM02-B1240000213	23.4°C	a day ago

Extra data

```
eth0, 10.0.3.1, nuc0-1.wilab1.ilabt.iminds.be, YK20823, /dev/ttyUSB0, ZOL-RM02-B1240000213, 234, glncwbfxambonm, Sun_Nov_11_07:28:17_CET_2018
```

	Node name	Result	YepkitID	Sensor device	Sensor name	Temperature	Date
+	nuc0-2	SUCCESS	YK20808	/dev/ttyUSB0	ZOL-RM02-B1240000215	24.3°C	22 days ago

Figure 43: w-iLab.t weekly testing of all nodes



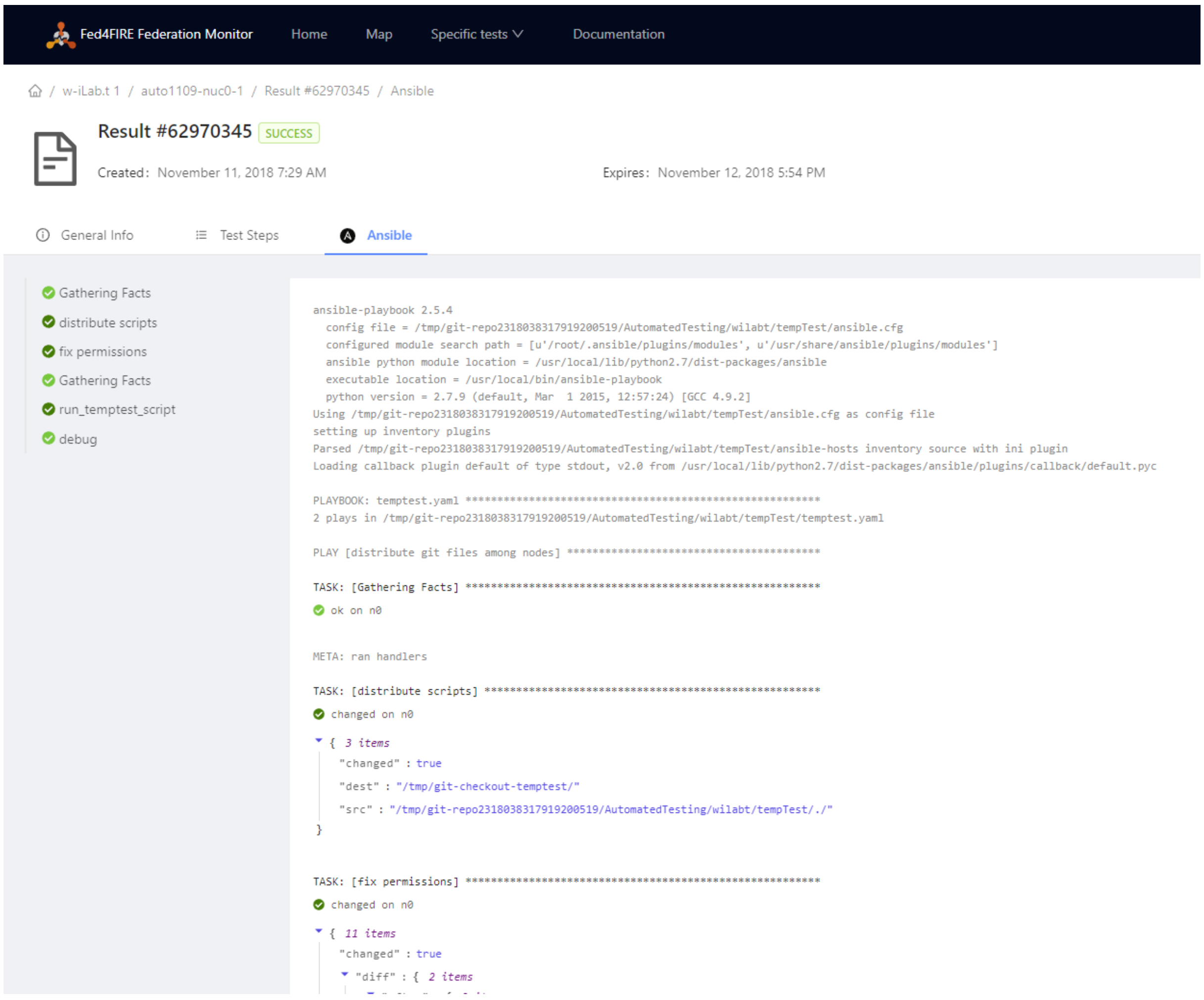
- nuc0-B1	FAILURE	YK20686	/dev/ttyUSB0	ZOL-RM02-B1002671

Extra data

```
eth0, 10.0.3.102, nuc0-B1.wilab1.ilabt.iminds.be, YK20686, /dev/ttyUSB0, ZOL-RM02-B1002671, ERROR, gxuycn5ckdfufw, Sun_Sep_30_11:44:12_CEST_2018
```

Figure 44: Extra info when clicking + at the left

Co-funded by the Horizon 2020 Framework Programme of the European Union



Result #62970345 SUCCESS

Created: November 11, 2018 7:29 AM

Expires: November 12, 2018 5:54 PM

```
ansible-playbook 2.5.4
  config file = /tmp/git-repo2318038317919200519/AutomatedTesting/wilabt/tempTest/ansible.cfg
  configured module search path = [u'/root/.ansible/plugins/modules', u'/usr/share/ansible/plugins/modules']
  ansible python module location = /usr/local/lib/python2.7/dist-packages/ansible
  executable location = /usr/local/bin/ansible-playbook
  python version = 2.7.9 (default, Mar  1 2015, 12:57:24) [GCC 4.9.2]
Using /tmp/git-repo2318038317919200519/AutomatedTesting/wilabt/tempTest/ansible.cfg as config file
setting up inventory plugins
Parsed /tmp/git-repo2318038317919200519/AutomatedTesting/wilabt/tempTest/ansible-hosts inventory source with ini plugin
Loading callback plugin default of type stdout, v2.0 from /usr/local/lib/python2.7/dist-packages/ansible/plugins/callback/default.pyc

PLAYBOOK: temptest.yaml ********************************************************
2 plays in /tmp/git-repo2318038317919200519/AutomatedTesting/wilabt/tempTest/temptest.yaml

PLAY [distribute git files among nodes] ****************************************

TASK: [Gathering Facts] ********************************************************
✓ ok on n0


META: ran handlers

TASK: [distribute scripts] *****************************************************
✓ changed on n0

▾ { 3 items
    "changed" : true
    "dest" : "/tmp/git-checkout-temptest/"
    "src" : "/tmp/git-repo2318038317919200519/AutomatedTesting/wilabt/tempTest/./"
}


TASK: [fix permissions] ********************************************************
✓ changed on n0

▾ { 11 items
    "changed" : true
  ▾ "diff" : { 2 items
```

Figure 45: Ansible output of w-iLab.t node test

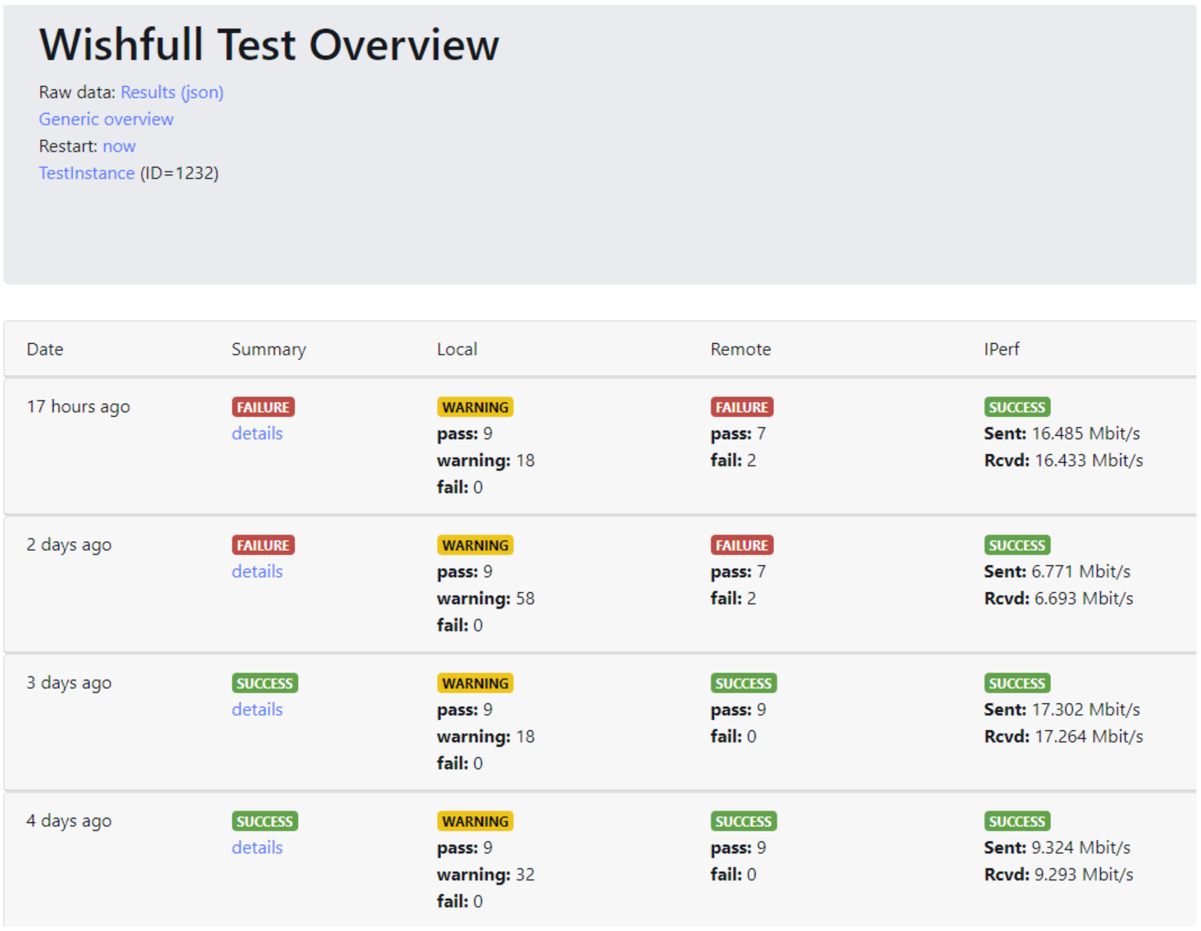
These tests are also running ansible scripts. In this way a testbed provider can provide us ansible scripts they want to run, and we provide the automation.

5.5.3 Automatically testing software on testbeds

For the Wishful project (http://www.wishful-project.eu/) , there was a demand to regularly test their software suite on real testbed nodes to verify if changes were still okay. The following tests are done once per day (Figure 46, https://flsmonitor.fed4fire.eu/wishfull/):

- Deploy local and global controller on a single node for simple basic functionality testing (column 'local')
- Deploy global controller and local controller on different nodes to test the messaging system over wifi (column 'remote')
- Deploy full scenario with access point and client and an iPerf performance test (column 'iPerf')

Co-funded by the Horizon 2020
Framework Programme of the European Union



Wishfull Test Overview

Raw data: Results (json)
Generic overview
Restart: now
TestInstance (ID=1232)

Date	Summary	Local	Remote	IPerf
17 hours ago	FAILURE details	WARNING **pass:** 9 **warning:** 18 **fail:** 0	FAILURE **pass:** 7 **fail:** 2	SUCCESS **Sent:** 16.485 Mbit/s **Rcvd:** 16.433 Mbit/s
2 days ago	FAILURE details	WARNING **pass:** 9 **warning:** 58 **fail:** 0	FAILURE **pass:** 7 **fail:** 2	SUCCESS **Sent:** 6.771 Mbit/s **Rcvd:** 6.693 Mbit/s
3 days ago	SUCCESS details	WARNING **pass:** 9 **warning:** 18 **fail:** 0	SUCCESS **pass:** 9 **fail:** 0	SUCCESS **Sent:** 17.302 Mbit/s **Rcvd:** 17.264 Mbit/s
4 days ago	SUCCESS details	WARNING **pass:** 9 **warning:** 32 **fail:** 0	SUCCESS **pass:** 9 **fail:** 0	SUCCESS **Sent:** 9.324 Mbit/s **Rcvd:** 9.293 Mbit/s

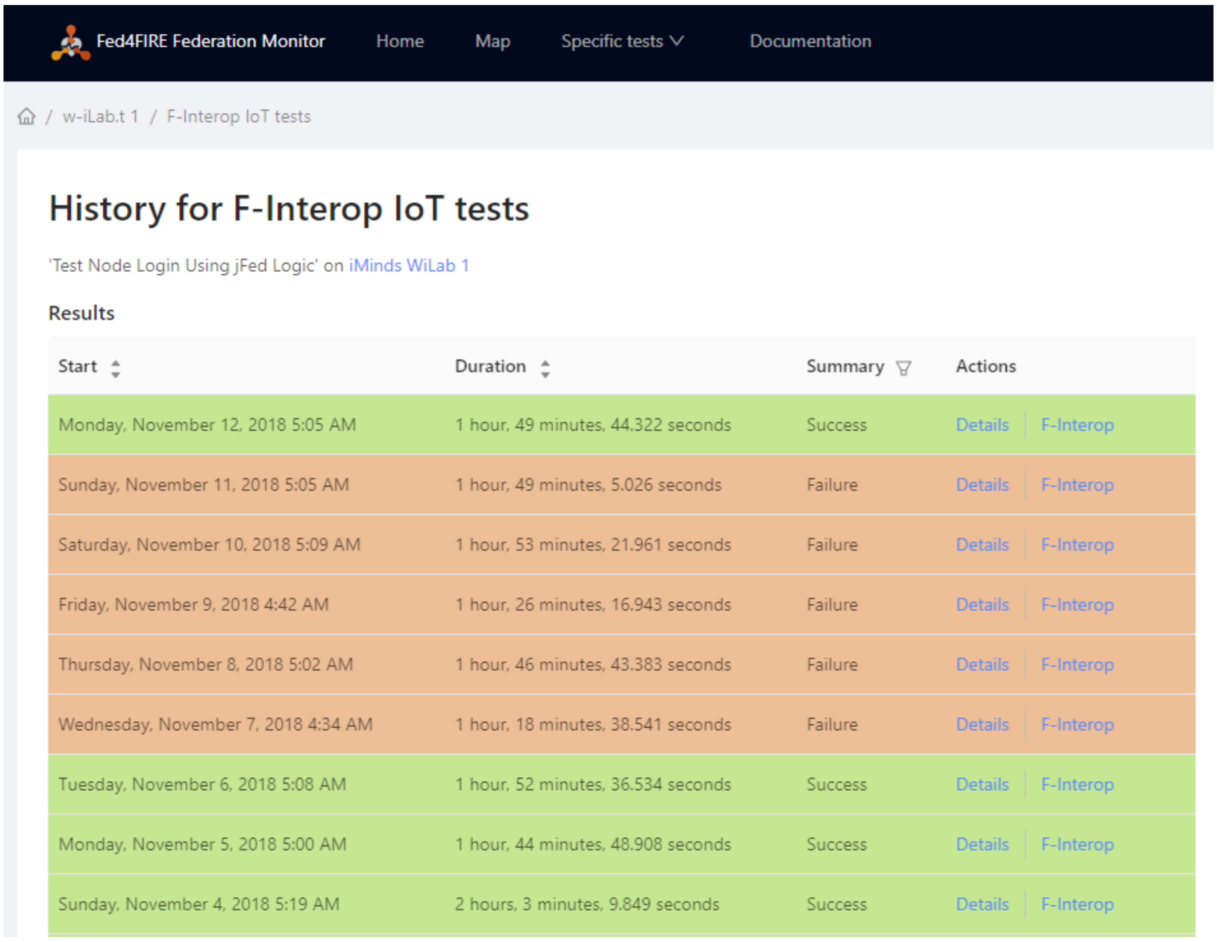
Figure 46: Wishful software testing, when clicking on 'details' more info on the test can be seen

5.5.4 Continuous IoT interop and conformance testing

In collaboration with the H2020 F-Interop project (https://www.f-interop.eu/), we have taken it even one step further. By deploying automatically the whole interop/conformance platform on a testbed and using testbed IoT devices, we can continuously run the F-interop test suites and verify interop and conformance on new versions of the IoT firmware, or new versions of the test suites.

Figure 47 shows an overview of the daily test run (https://fedmon.fed4fire.eu/history/1413). When clicking on 'F-interop' , one can see Figure 48 with details on all the tests for multiple pairs of testing.

Co-funded by the Horizon 2020 Framework Programme of the European Union



History for F-Interop IoT tests

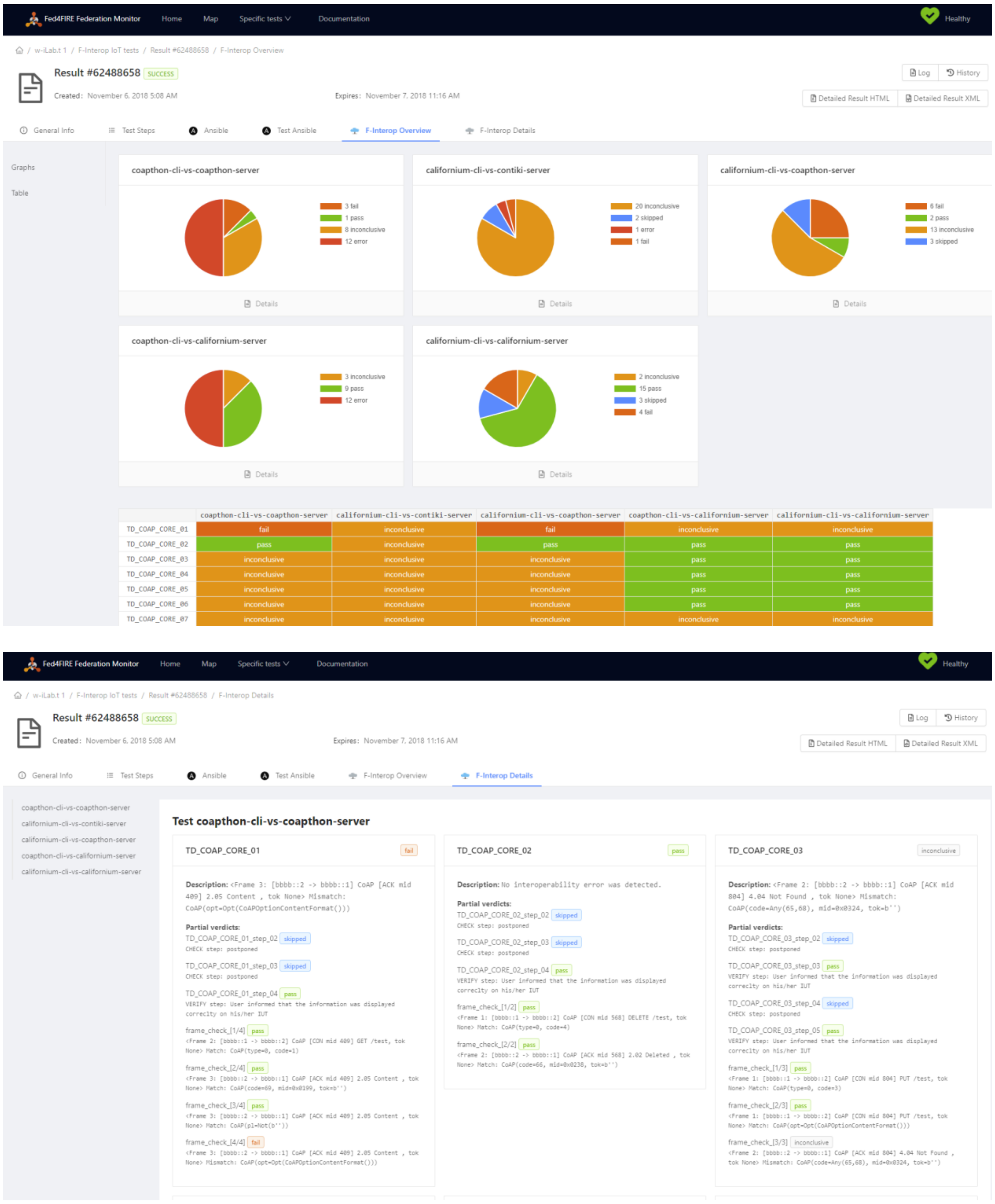
'Test Node Login Using jFed Logic' on iMinds WiLab 1

Results

Start	Duration	Summary	Actions
Monday, November 12, 2018 5:05 AM	1 hour, 49 minutes, 44.322 seconds	Success	Details | F-Interop
Sunday, November 11, 2018 5:05 AM	1 hour, 49 minutes, 5.026 seconds	Failure	Details | F-Interop
Saturday, November 10, 2018 5:09 AM	1 hour, 53 minutes, 21.961 seconds	Failure	Details | F-Interop
Friday, November 9, 2018 4:42 AM	1 hour, 26 minutes, 16.943 seconds	Failure	Details | F-Interop
Thursday, November 8, 2018 5:02 AM	1 hour, 46 minutes, 43.383 seconds	Failure	Details | F-Interop
Wednesday, November 7, 2018 4:34 AM	1 hour, 18 minutes, 38.541 seconds	Failure	Details | F-Interop
Tuesday, November 6, 2018 5:08 AM	1 hour, 52 minutes, 36.534 seconds	Success	Details | F-Interop
Monday, November 5, 2018 5:00 AM	1 hour, 44 minutes, 48.908 seconds	Success	Details | F-Interop
Sunday, November 4, 2018 5:19 AM	2 hours, 3 minutes, 9.849 seconds	Success	Details | F-Interop

Figure 47: Daily F-interop platform testing

Co-funded by the Horizon 2020
Framework Programme of the European Union



	coapthon-cli-vs-coapthon-server	californium-cli-vs-contiki-server	californium-cli-vs-coapthon-server	coapthon-cli-vs-californium-server	californium-cli-vs-californium-server
TD_COAP_CORE_01	fail	inconclusive	fail	inconclusive	inconclusive
TD_COAP_CORE_02	pass	inconclusive	pass	pass	pass
TD_COAP_CORE_03	inconclusive	inconclusive	inconclusive	pass	pass
TD_COAP_CORE_04	inconclusive	inconclusive	inconclusive	pass	pass
TD_COAP_CORE_05	inconclusive	inconclusive	inconclusive	pass	pass
TD_COAP_CORE_06	inconclusive	inconclusive	inconclusive	pass	pass
TD_COAP_CORE_07	inconclusive	inconclusive	inconclusive	inconclusive	inconclusive

Fed4FIRE Federation Monitor
Home
Map
Specific tests
Documentation
Healthy
w-iLab.t 1 / F-Interop IoT tests / Result #62488658 / F-Interop Details
Result #62488658 SUCCESS
Created: November 6, 2018 5:08 AM
Expires: November 7, 2018 11:16 AM
Log
History
Detailed Result HTML
Detailed Result XML
General Info
Test Steps
Ansible
Test Ansible
F-Interop Overview
F-Interop Details
coapthon-cli-vs-coapthon-server
californium-cli-vs-contiki-server
californium-cli-vs-coapthon-server
coapthon-cli-vs-californium-server
californium-cli-vs-californium-server

Test coapthon-cli-vs-coapthon-server

TD_COAP_CORE_01 fail

Description: `<Frame 3: [bbbb::2 -> bbbb::1] CoAP [ACK mid 409] 2.05 Content , tok None> Mismatch: CoAP(opt=Opt(CoAPOptionContentFormat()))`

Partial verdicts:

TD_COAP_CORE_01_step_02 skipped
```
CHECK step: postponed
```
TD_COAP_CORE_01_step_03 skipped
```
CHECK step: postponed
```
TD_COAP_CORE_01_step_04 pass
```
VERIFY step: User informed that the information was displayed
correclty on his/her IUT
```
frame_check_[1/4] pass
```
<Frame 2: [bbbb::1 -> bbbb::2] CoAP [CON mid 409] GET /test, tok
None> Match: CoAP(type=0, code=1)
```
frame_check_[2/4] pass
```
<Frame 3: [bbbb::2 -> bbbb::1] CoAP [ACK mid 409] 2.05 Content , tok
None> Match: CoAP(code=69, mid=0x0199, tok=b'')
```
frame_check_[3/4] pass
```
<Frame 3: [bbbb::2 -> bbbb::1] CoAP [ACK mid 409] 2.05 Content , tok
None> Match: CoAP(pl=Not(b''))
```
frame_check_[4/4] fail
```
<Frame 3: [bbbb::2 -> bbbb::1] CoAP [ACK mid 409] 2.05 Content , tok
None> Mismatch: CoAP(opt=Opt(CoAPOptionContentFormat()))
```

TD_COAP_CORE_02 pass

Description: `No interoperability error was detected.`

Partial verdicts:

TD_COAP_CORE_02_step_02 skipped
```
CHECK step: postponed
```
TD_COAP_CORE_02_step_03 skipped
```
CHECK step: postponed
```
TD_COAP_CORE_02_step_04 pass
```
VERIFY step: User informed that the information was displayed
correclty on his/her IUT
```
frame_check_[1/2] pass
```
<Frame 1: [bbbb::1 -> bbbb::2] CoAP [CON mid 568] DELETE /test, tok
None> Match: CoAP(type=0, code=4)
```
frame_check_[2/2] pass
```
<Frame 2: [bbbb::2 -> bbbb::1] CoAP [ACK mid 568] 2.02 Deleted , tok
None> Match: CoAP(code=66, mid=0x0238, tok=b'')
```

TD_COAP_CORE_03 inconclusive

Description: `<Frame 2: [bbbb::2 -> bbbb::1] CoAP [ACK mid 804] 4.04 Not Found , tok None> Mismatch: CoAP(code=Any(65,68), mid=0x0324, tok=b'')`

Partial verdicts:

TD_COAP_CORE_03_step_02 skipped
```
CHECK step: postponed
```
TD_COAP_CORE_03_step_03 pass
```
VERIFY step: User informed that the information was displayed
correclty on his/her IUT
```
TD_COAP_CORE_03_step_04 skipped
```
CHECK step: postponed
```
TD_COAP_CORE_03_step_05 pass
```
VERIFY step: User informed that the information was displayed
correclty on his/her IUT
```
frame_check_[1/3] pass
```
<Frame 1: [bbbb::1 -> bbbb::2] CoAP [CON mid 804] PUT /test, tok
None> Match: CoAP(type=0, code=3)
```
frame_check_[2/3] pass
```
<Frame 1: [bbbb::1 -> bbbb::2] CoAP [CON mid 804] PUT /test, tok
None> Match: CoAP(opt=Opt(CoAPOptionContentFormat()))
```
frame_check_[3/3] inconclusive
```
<Frame 2: [bbbb::2 -> bbbb::1] CoAP [ACK mid 804] 4.04 Not Found ,
tok None> Mismatch: CoAP(code=Any(65,68), mid=0x0324, tok=b'')
```

Figure 48: F-interop COAP interop testing automatically on Fed4FIRE testbeds

Co-funded by the Horizon 2020
Framework Programme of the European Union

6 INTERCONNECTIVITY

For interconnectivity, jFed was extended in two ways:

- A better visual way of showing the different connectivity methods. Figure 49 gives an example of all methodes:
  - Normal layer 2 connectivity (link1)
  - Tagged vlan connectivity (link0)
  - GRE tunnel (link2 and link3)
  - Stitched vlans between multiple testbeds (link4)
- Where possible, chose automatically the most default way. E.g. for City of Things, no vlans are possible, no stitching, so by default a gre tunnel is chosen (link3). This will help experimenters to easily interconnect in the 'right' way (seen as most adequate by the testbed owners)

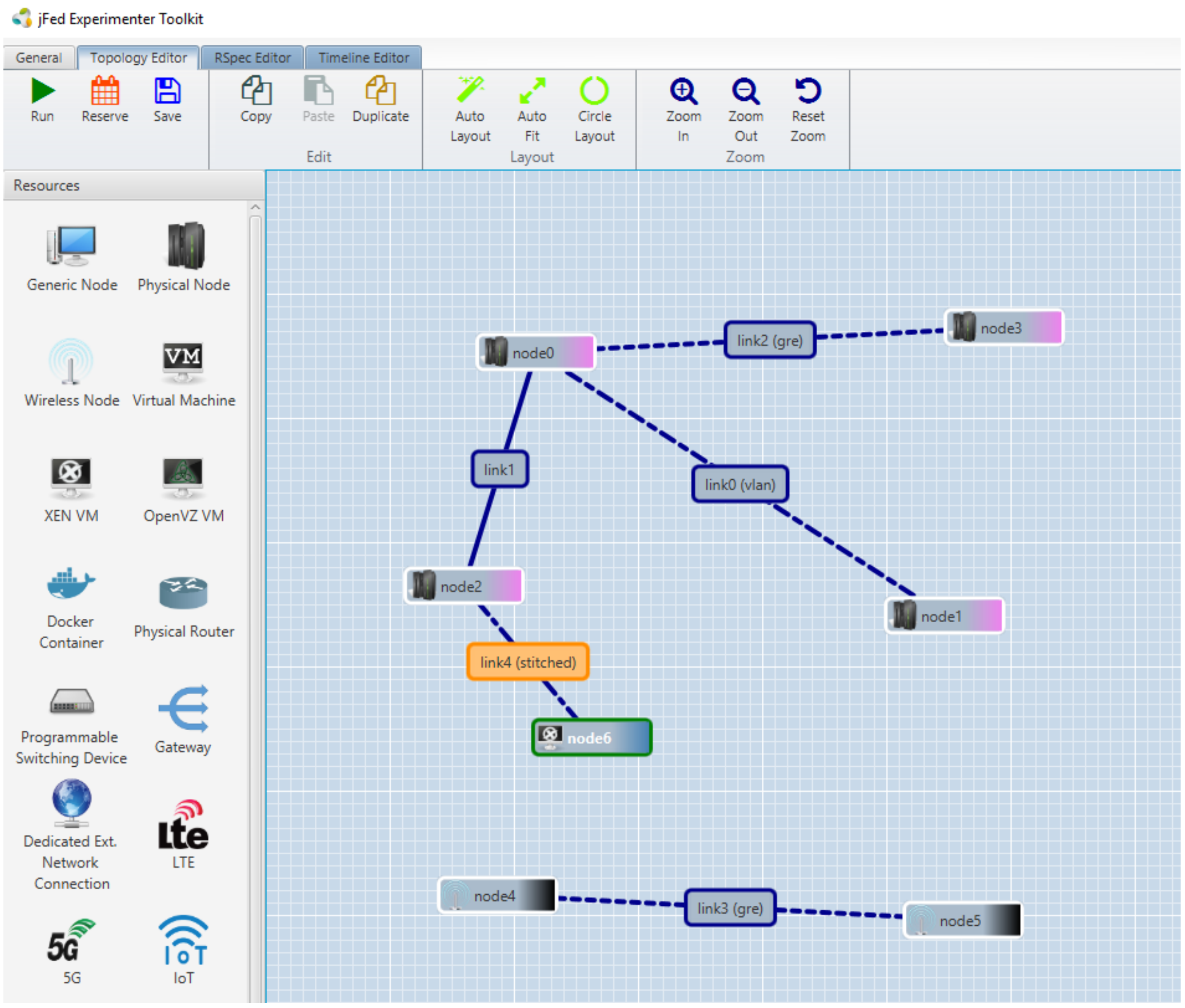


Figure 49: jFed visualisation of different interconnectivity methods

Co-funded by the Horizon 2020 Framework Programme of the European Union

7 SERVICE ORCHESTRATION (YOUREPM)

YourEPM is a tool that was started in Fed4FIRE. Its goal is to have complex experiment orchestration/workflows in the federation. In the first 18 months of this project, not that much work was spent on the implementation yet, besides the identification of the topics/requirements to work on in the next period (see also D3.1):

- Multi-tenant use
- Authentication of users
- Management of 3rd party access to the tool
- Integration with jFed
- Creation of complex experiments as example

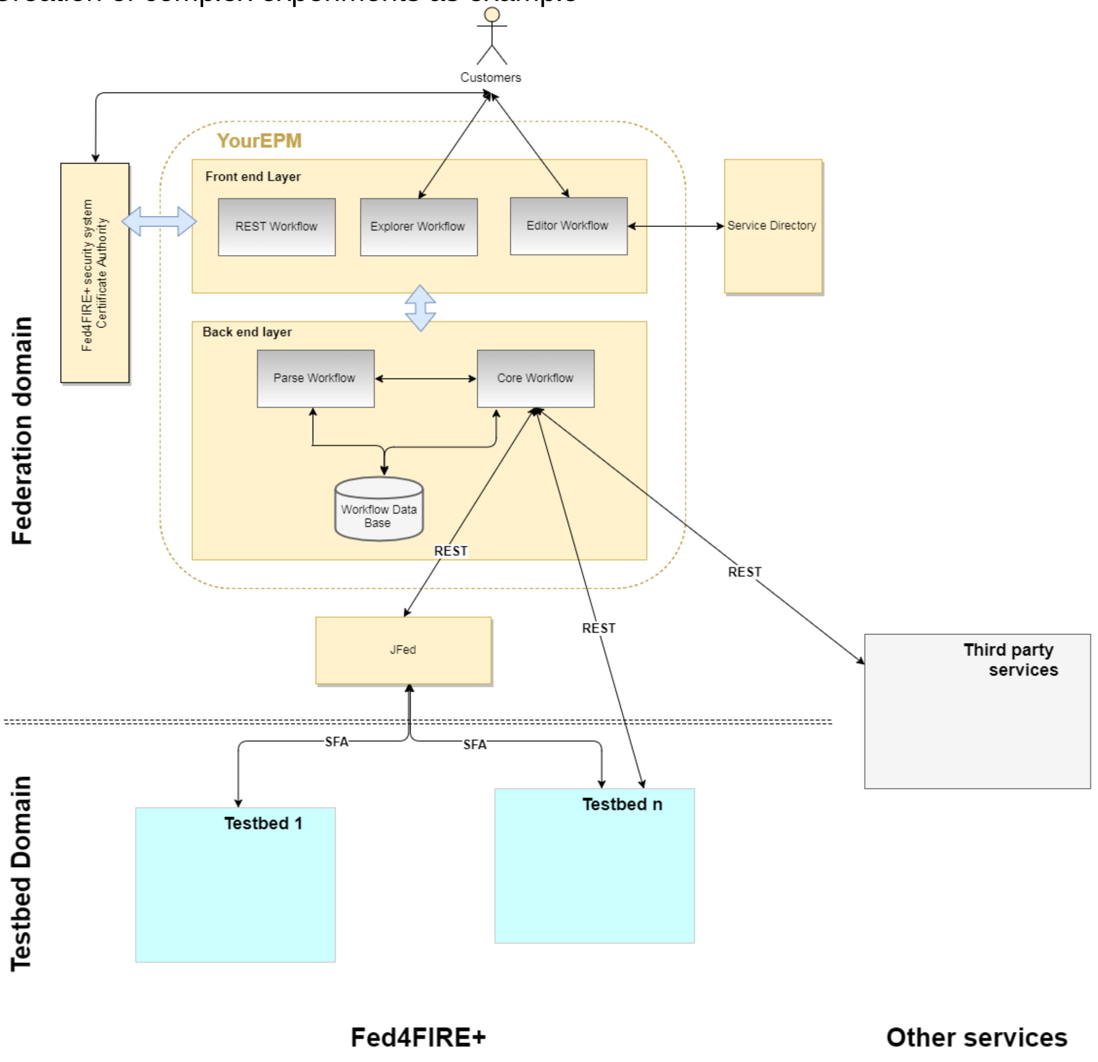


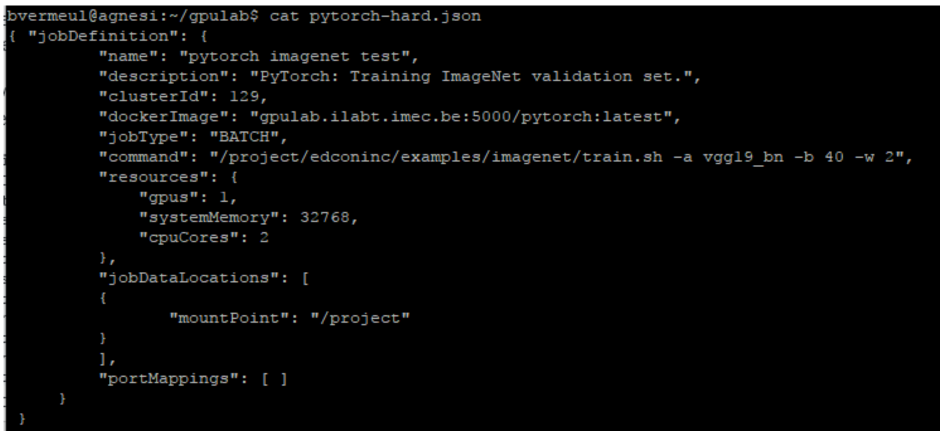
Figure 50: YourEPM architecture

Co-funded by the Horizon 2020 Framework Programme of the European Union

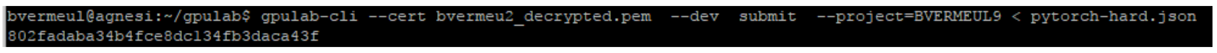
8 AUTHENTICATION PROXY SERVICE

The authentication proxy service for which the requirements are listed in D3.1 is implemented and tested for a service called GPULab (https://doc.ilabt.imec.be/ilabt-documentation/gpulab.html). GPULab offers easy GPU based experimentation based on docker containers and Fed4FIRE accounts.

For this service, users can use a client (Figure 51) which talks through REST to the authentication proxy, which talks both to the Fed4FIRE authority and to the specific GPULab service. We will report in more detail on this in D3.04 (developments of the 2nd cycle).



```
bvermeul@agnesi:~/gpulab$ cat pytorch-hard.json
{ "jobDefinition": {
         "name": "pytorch imagenet test",
         "description": "PyTorch: Training ImageNet validation set.",
         "clusterId": 129,
         "dockerImage": "gpulab.ilabt.imec.be:5000/pytorch:latest",
         "jobType": "BATCH",
         "command": "/project/edconinc/examples/imagenet/train.sh -a vgg19_bn -b 40 -w 2",
         "resources": {
             "gpus": 1,
             "systemMemory": 32768,
             "cpuCores": 2
         },
         "jobDataLocations": [
         {
                "mountPoint": "/project"
         }
         ],
         "portMappings": [ ]
     }
 }
```



```
bvermeul@agnesi:~/gpulab$ gpulab-cli --cert bvermeu2_decrypted.pem  --dev  submit  --project=BVERMEUL9 < pytorch-hard.json
802fadaba34b4fce8dc134fb3daca43f
```

Figure 51: GPULab client

Co-funded by the Horizon 2020
Framework Programme of the European Union

9 CENTRAL BROKER

9.1 INTRODUCTION

The Central Broker is a component (Figure 52) that will help users in selecting resources, by 'brokering' between all the testbed resources and the users.

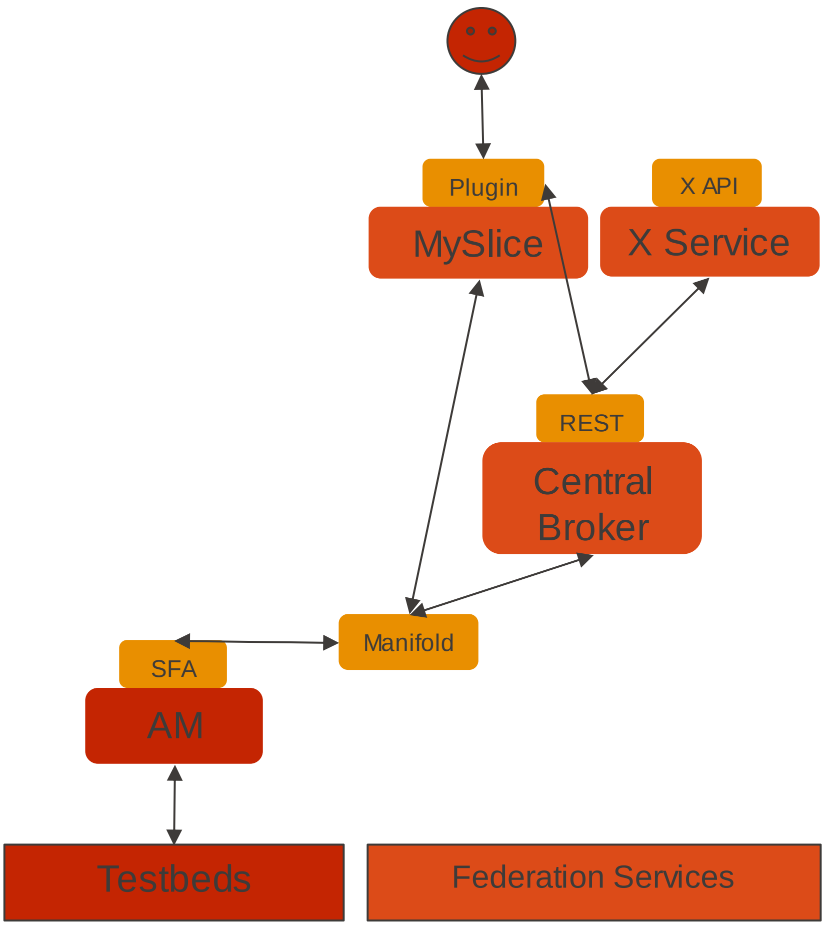


Figure 52: Central Broker

9.2 REDESIGN

During the first cycle of developments for Fed4FIRE+, a major architectural transition has occurred for the central broker; the release of MySlice version 2. The new MySlice provides a more robust and agile back-end that can be used by the Central Broker towards providing a better user experience. CERTH has redesigned the overall architecture of Central Broker to support this new version.

Co-funded by the Horizon 2020 Framework Programme of the European Union

Although, since currently there is no MySlice v2 instance in production for Fed4FIRE+, CERTH has implemented all the additions that are being reported in the current document, in a sandbox environment using pre-captured data and not live data.

For future cycles CERTH will integrate the Central Broker with MySlice version 2, as soon as a production instance is offered for Fed4FIRE+.

9.3 NEW CAPABILITIES

CERTH further enhanced the capabilities of the Central Broker to support quota policies per domain (testbed users) as well as per urn (specific user). Quotas are related to number of resources and maximum time of reservation of these resources. Enabling/disabling policies can be done with a configuration file. With quota policies enabled, Central Broker has the ability to take into consideration the quotas before suggesting the topology to the experimenter.

9.4 NEW TESTBED SUPPORT

A crucial factor for rendering the Central Broker into a useful federation service for the experimenters, is its capability in supporting many and different kinds of testbeds. During the first cycle of developments a prototype that enables Central Broker to include w-iLabt pool of resources has been implemented.

9.5 NEW MYSLICE – VERSION 2

The first prototype of MySlice v1 was tightly coupled with the underlying AM API. As long as one of the Aggregate Managers of the federation was down or answered with a long delay, the user experience was affected. Moreover, a large number of redundant queries were sent at the same time by different users, such as the list of resources. The version 2 of MySlice addresses those issues improving thus the user experience.

The new architecture is composed of 5 layers with a clear separation of concerns: Web frontend, APIs (REST/WS), Database, Services with workers and Library (XML-RPC), see Figure 53.

The frontend has been redesigned using the ReactJS framework. The benefit of using such a framework is to create generic components that can be re-use in different views depending on the properties passed to the components. Moreover, the management of a store that maintains a state of a component or a view is very well suited for an event-oriented application.

Co-funded by the Horizon 2020
Framework Programme of the European Union

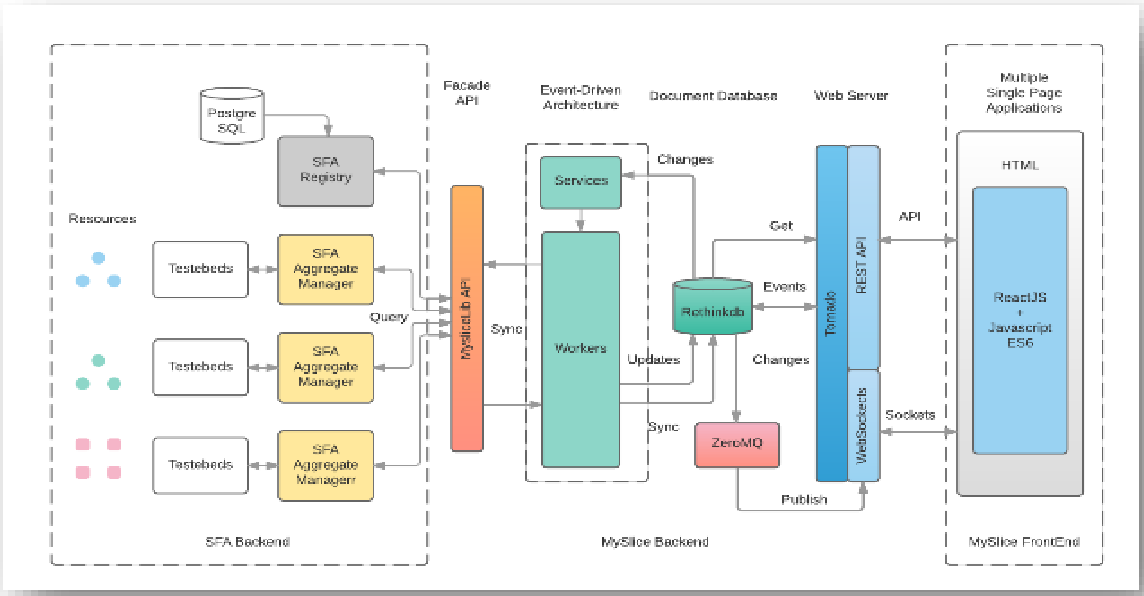


Figure 53: MySlice v2 architecture

We have clearly defined the REST and WebSocket APIs used by the React components and third-party software. The web components are able to get or post data through the REST API and can be notified of a change through the WebSocket, providing a very interactive frontend. Some interactions of the user with the frontend are generating events that are stored in a document oriented database. The MySlice router is then responsible to place these events in the relevant queue depending on their type. Each type of event is asynchronously processed by a service. The services are calling workers that can be multithreaded to scale up the capabilities of the system. The workers are responsible of the interactions with the distributed testbeds through the AM API (XML-RPC) and with the SFA Registry, which is the root authority of the federation providing the credentials to access the testbeds.

Co-funded by the Horizon 2020 Framework Programme of the European Union

10 ONTOLOGIES: USING SEMANTIC WEB TECHNOLOGIES TO QUERY AND MANAGE INFORMATION WITHIN FEDERATED CYBER-INFRASTRUCTURES

10.1 INTRODUCTION

This section is largely based on the following publication: Willner, A.; Giatili, M.; Grosso, P.; Papagianni, C.; Morsey, M.; Baldin, I. Using Semantic Web Technologies to Query and Manage Information within Federated Cyber-Infrastructures. *Data* 2017, *2*, 21. It is based on work of Fed4FIRE, Fed4FIRE+ and other projects. It gives a complete overview of the base ontology work for federation of testbeds. The references can be found in 10.8.

Cloud computing supports many distributed applications that are vital to the economy and to the advancement of science. The rising popularity of cloud computing and the diversity of available resources create an urgent need to match those resources optimally to the requests of end-users.

The desired level of self-serve operation within the cloud obviates the need for intervention by IT departments, allowing end-users direct and independent access to the computational infrastructure. As a result, end-users can deploy the necessary infrastructure and software solutions rapidly. Toward this end, accurate modeling of the infrastructure must support abstract representation of various resources, simplify interactions with them, and expose the right levels of information. The next frontier in cloud computing lies in supporting widely distributed clouds and the various aspects of the architectures needed to manage resources across multiple administrative domains. These problems are also closely related to future Internet research in academia, as well as to emerging commercial technologies like the Internet of Things (IoT) [1].

Modeling cloud infrastructures in a manner that supports effective matching of users' requests with available resources is a challenging task. The issue becomes even more complex in the context of distributed cloud systems with multiple infrastructure owners. In the academic research the same problem is encountered when trying to describe computational resources, scientific instruments and testbeds, which belong to different institutions and must be used by inter-disciplinary and inter-institutional collaborative teams. In such environments each infrastructure owner may model resources using their particular information and data modeling approach to set up the system quickly and attract users. The end-result, however, is user lock-in and inability to easily leverage available resources if they belong to different owners. Thus, resource matching and recommendation *based on common models* becomes of great importance.

This section describes Open-Multinet (OMN) [2], a set of ontologies that rely on Semantic Web (Semantic Web) [3] technologies. It was designed by an international team of academic researchers who are intimately familiar with the related problems. The OMN researchers are also involved in multiple efforts to design a federation of Future Internet and cloud testbeds spanning the US and the EU, to be used for at-scale experimentation with novel concepts in networking and distributed systems. While we briefly introduced the ontology set in [2] and

Co-funded by the Horizon 2020 Framework Programme of the European Union

presented a preliminary description of its application in the context of a federated cloud environments in [4], in this section we complement our previous work (of Fed4FIRE) by an extended description of the OMN ontology set and we further added new evaluation results of the overall OMN framework (this is done in the Fed4FIRE+ project).

Motivation for our work comes largely from our experience with the growth of academic networking, including the proliferation of cloud testbeds. Their ad hoc attempts to federate with each other, i.e., to make their resources available to wider communities of users through common interfaces, suffer from a lack of common models to describe available resources. Testbed owners use such models chiefly to provide their users with information about available resources, e.g., compute nodes, storage, network switches, and wireless access points. Each user, in turn, employs similar models to request resources from the testbeds, describing in some detail the exact configuration of available resources needed from the testbed.

Most testbeds are small when they first launch. Their designers often spend little time thinking through the information model that they wish to use to present resource information to users. Testbeds frequently rely on home-brewed solutions utilizing syntactic schema specifications serialized using XML or JSON, sometimes referred to as *RSpecs*, although RSpec is also a name of a specific XML dialect used by most of the testbeds in the US and EU. Documents expressed in those languages are passed between the users and the testbed management software in order to describe the available resources and to request specific configurations of resources for the experiments. While the built-in mechanisms in those languages allow for straightforward verification of document syntax, few mechanisms are available for validation of semantic correctness. These solutions typically rely on structure-implied semantics to validate correctness by associating semantic meaning rigidly with the position of information elements within the document.

These approaches tend to work in early phases of the design. As the diversity of resources grows, however, and as the sophistication of users increases, the need arises for extension mechanisms. Demand emerges for more powerful resource descriptions. The extension mechanisms then inevitably relax the structure-implied semantics, thus making validation of documents progressively more difficult. We observed this development first-hand in the case of US Global Environment for Network Innovations (GENI) [5] and EU Future Internet Research and Experimentation (FIRE) [6] testbed-federation efforts. XML schema extensions were introduced to allow different federation members to describe the unique attributes of their cloud testbeds. The extensions, we found, made it possible to create syntactically valid but semantically invalid documents requesting resources from a testbed, e.g., by requesting that a particular operating-system image be attached to a network interface instead of to a compute node.

Informed by these experiences, we decided to adopt Semantic Web technologies, which provided us with a number of advantages:

- A common standardized model is used to describe cloud and testbed infrastructures. The extensibility of this model is built into it from the start in the form of additional ontologies that describe new types of resources. The machinery to deal with extensions is built into standard semantic web toolkits, leaving the designers free to think about the information model while affixing the data model.

- Different resources and descriptions easily can be related and connected semantically. Semantic web mechanisms intuitively represent computer network graph structures. Network topologies are embedded into the RDF graph using graph homeomorphisms and then are annotated with additional information, addressing *structural and semantic constraints* in a single structure.
- Model errors can be detected early, before testbed resources are provisioned, by using many standard inference tools.
- Rules can in particular be used to complement queries. Rules for harmonizing relationships should to be defined and applied on the federation level. This is where specialties and commonalities of the involved testbeds are known and this approach lifts the burden from users to formulate complex queries.
- The annotation process, i.e., the conversion from XML-based RSpecs to RDF-based graphs, is automatic and configurable to take testbed specific extensions and federation-wide agreements into account.
- Using standard Semantic Web tools, complex queries can be formulated to discover resources. A common way for testbeds to operate is by ingesting JSON/XML or other encoding of the user request or resource advertisement and then converting it into a non-portable native form on which queries and embeddings are performed. Semantic web tools allow us to store testbed-state information natively in RDF and to operate on that information using a multitude of native inference and query tools, thus simplifying and abstracting many parts of testbed operations.
- Once cloud resources are described semantically, they can be interlinked to other Linked Open Data (LOD) [7] cloud data sets. These linkages provide additional information about resource availability or constraints and help to link resources, e.g., to policies governing their allocation.
- Semantic resource descriptions support convergence from multiple syntactic-schema based representations of testbed resources to a single semantically enriched representation that combines information from multiple sources. Such sources include various RSpecs describing testbed resources, out-of-band knowledge that may be encoded in resource names or contained in human-readable online Web pages, an approach consistent with Ontology-based Data Access (OBDA). Encoding this information in a structured way into a single representation prepares it for direct analysis, without need of an intermediate representation. Answers are derived by matching resources required by the user to those available at one or more different testbeds, federating the testbeds automatically, with minimal human intervention.

We believe that our approach represents an interesting application of OBDA to a novel area of use that combines information search and retrieval with active infrastructure-resource management.

The OMN development effort consisted of several phases, starting with the upper ontology design, followed by the design of several critical subordinate ontologies for, e.g., resource monitoring. We relied heavily on previous work in this area, directly incorporating, for instance, the Infrastructure and Network Description Language (INDL) [8,9] ontology for describing computer networks and the Networking innovations Over Virtualized Infrastructures (NOVI) [10] ontology for federated infrastructures. We then started developing tools that utilize the ontology, including converters from the various testbed resource description formats to OMN, inference rules for knowledge extension to complement the conversion process, and rule sets for semantic validation of the documents. We also developed standard queries that assist the testbed resource-matching algorithms in extracting needed information from the testbed resource descriptions.

The remainder of the section is structured as follows. We give a brief overview of related work in the context of (federated) heterogeneous computing infrastructures in Section 10.2. In Section 10.3, we present the OMN ontology set. Section 10.4 shows how we extract information from RSpecs and annotate it using additional knowledge extraction from out-of-band information. Querying and validation using traditional semantic web stools are then performed by the tools built on this framework. Section 10.5 shows the performance and applicability of our tools. Finally, we close in Section 10.6 with conclusions, considerations, and a description of future work. Section 10.7 has a list of abbreviations and 10.8 a list of references. The last section 10.9 describes how this will be used in matchmaking capabilities for WP4.

10.2 RELATED WORK

Many application disciplines shifted the focus from tree-based data models (e.g., XML-based syntactic schemas) to semantic models. This change is reflected in development of ontologies to support, for example, the operation of Grids, Clouds, and now the Internet of Things. These efforts have informed our own OMN development. In the coming section, we provide an overview of these efforts.

10.2.1 Semantic Models For Grids, Clouds and IoT

In the context of Grid Computing, the Grid Laboratory for a Uniform Environment (GLUE) [11] schema was started 15 years ago to support interoperability between Grid projects by defining a schematic vocabulary with serializations to XML, LDAP, and SQL [12]. A lack of formalism and a consequent inability to perform logical operations on data motivated the transition to Semantic Open Grid Service Architecture (S-OGSA) [13].

Semantic Web service discovery [14,15] addresses the automated discovery of Web services that satisfy given requirements. The discovery process uses a matchmaking algorithm to find potential Web services that might solve the problem at hand. Such methods, however, are inadequate to handle the complex interconnected computing infrastructures addressed by our work. Research on matching concentrates mainly on Web services [16], specifically, on semantic similarities between input and output parameters of various services. Our resource-matching involves more than matching available resources to the requirements of the end-user. We also need to identify homeomorphic embeddings of requested topologies within available resource topologies. The combination of such semantic and structural constraints leads to a substantially greater challenge. Pedrinaci et al. introduced Linked USDL (Linked USDL) [17], a vocabulary that applies research conducted on Semantic Web services to USDL [18,19]. Linked USDL provides comprehensive means to describe services in support of automated processing. It focuses only on services, and is unsuited to the description of cloud

Co-funded by the Horizon 2020
Framework Programme of the European Union

infrastructures. Ontologies such as the Semantic Markup for Web Services (OWL-S) [20] or Good Relations (GR) [21], however, are of interest to our work, and are referenced in part in our ontology.

In the domain of Cloud Computing, researchers are working to ensure interoperability on a semantic level. Since 2008, work has progressed in the development of ontologies and of tools for semantic cloud computing [22–24]. Haase et al. [25], for example, introduced an approach to administration of enterprise cloud environments, using semantic Web technologies. They proposed a Semantic Web-based product called eCloudManager, which incorporates an ontology to model its cloud data. However, the system and its ontology only focus on the management aspect of cloud systems, and the data are not open for usage. In another example, Haak et al. [26] proposed an ontology-based optimization methodology that enables cloud providers to detect the best resource set to satisfy a user's request. Their framework handles only a single administrative domain, whereas we seek to cover a distributed set of provider domains.

A paradigm shift is in progress in favor of Intercloud Computing. For instance, 20 approaches to this new challenge are presented in [27]. Within this context, Manno et al. proposed the use of the semantic Federated Cloud Framework Architecture (FCFA) [28] to manage resource life cycles based on formal models. In contrast, the Intercloud architecture developed within the Institute of Electrical and Electronics Engineers (IEEE) Standard for Intercloud Interoperability and Federation (P2302) [29,30] Working Group uses graphs to describe and to discover cloud resources based on the existing Open-Source API and Platform for Multiple Clouds (mOSAIC) [31] ontology. Both approaches are being considered as domain-specific extensions to our work. In addition, Santana-Pérez et al. [32] proposed a scheduling algorithm that was suitable for federated hybrid cloud systems. The algorithm applies semantic techniques to scheduling and to matching tasks with the most suitable resources. The information model is based on the Unified Cloud Interface (UCI) project ontologies, which cover a wide range of details but which cannot handle Intercloud systems. Le and Kanagasabai [33,34] also proposed ontology-based methodologies to discover and to broker cloud services. They use Semantic Web technologies for user requirements and for cloud provider advertisements, and then apply an algorithm to match each requirement list to advertised resource units. Multiple levels of matching are defined, ranging from an exact match to no match. These methodologies concentrate only on Infrastructure as a Service (IaaS) provisioning. Moreover, they typically neither export their data nor provide a SPARQL Protocol And RDF Query Language (SPARQL) [35] endpoint, thereby hindering reuse of and access to data.
Interest has soared recently in the uses and challenges of the Internet of Things in which many heterogeneous devices from different administrative domains communicate with each other. Semantic models are needed for the IoT. The European Research Cluster on the Internet of Things (IERC) has established Activity Chain 4—Service Openness and Interoperability Issues/Semantic Interoperability (AC4) [36], and semantic models such as the Semantic Sensor Network (SSN) [37] ontology have been developed. Support for semantics in Machine-To-Machine Communication (M2M) [38] has received further attention [39]. The primary applicable standardization activity from the European Telecommunications Standards Institute (ETSI) M2M Working Group has identified the need for a semantic resource descriptions in [40]. The successor, oneM2M (http://onem2m.org) [41], already has established the OneM2M Working Group 5 Management, Abstraction and Semantics (MAS). With the recent establishment of the World Wide Web Consortium (W3C) Web of Things (WoT) [42] Working Group, semantic vocabularies will be developed to describe data and interaction models.

Co-funded by the Horizon 2020 Framework Programme of the European Union

10.2.2 OMN Background

Development of our approach, the OMN ontology set, started within the Federation for FIRE (Fed4FIRE) [43] project. The aim was to extend and to harmonize related work for the FIRE initiative, which has been developed within the context of federated networks and e-infrastructures. Our main motivation was the state of the art in the Future Internet (FI) experimentation context, which considers only simple schema-based models. The Slice-based Federation Architecture (SFA) [44] is the de-facto standard Application Programming Interface (API) for testbed federation. It uses XML-based RSpecs to describe, to discover, and to manage resources in a simple declarative way. However, it cannot support complex queries combining structural and semantic constraints or knowledge analysis. OMN ontology design reuses concepts previously defined in RSpecs, but also leverages significant prior efforts to define ontologies targeting cyber-infrastructure management.

The Open Grid Forum (OGF) Network Mark-Up Language (NML) [45] is a well established ontology for modeling computer networks. It provides a framework for definition and description of topologies ranging from simple networks comprising a few nodes and connections to massive, complex networks with hundreds or thousands of nodes and links. The model underwent a thorough review and definition process, finally becoming an OGF standard. While NML lacks concepts and properties required to describe federated infrastructures, OMN adopts NML in order to model the networking aspects of the infrastructure.

In comparison with NML, the INDL addresses virtualization of resources and services. It supports description, discovery, modeling, composition, and monitoring of those resources and services. The INDL actually imports NML to describe attached computing infrastructures in a manner that is independent of technology and vendor. It offers the capacity to extend coverage to emerging network architectures. The INDL, however, does not support infrastructure federation, in which several different testbeds are interconnected experimentally.

Semantic models developed within the European NOVI and GEYSERS [46] projects have been used to describe federated infrastructures, user requests, policies, and monitoring information. They also support virtualization concepts for computing and networking devices. They have been adopted by OMN where their incorporation is appropriate. In the first project, the proposed Information Modeling Framework (IMF) [47] represents resources from the same or from different infrastructure providers.

In parallel to this, within the GENI initiative, the Network Description Language based on the Web Ontology Language (NDL-OWL) [48–51] model specifies capabilities to control and to manage complex networked testbed infrastructures. Lessons learned from live deployments of NDL-OWL in GENI proved informative in OMN modeling discussions.

Efforts related to describing APIs via OWL-S or DAML-S are not applicable directly to our problem, since they focus on the description of Web-service APIs. The testbeds in our research converged on a set of simple API calls like *createSlice* (requesting that a desired network topology be created) or *listResources* (requesting information about available infrastructure resources). The complexity lies in the parameters passed in those calls, not in the diversity of types of parameters serving as inputs or outputs to the APIs. Those parameters often are represented by XML documents describing requested or available testbed resource topologies. Our goal is to replace those syntactic-schema-based representations with

Co-funded by the Horizon 2020
Framework Programme of the European Union

semantic-web based views. We want them to include enough information to support native querying based on both structural and semantic constraints, either by the users or by testbed management algorithms.

10.3 OPEN-MULTINET ONTOLOGY SET

Following Noy and McGuiness [52], the first step for defining a formal information model is to determine the specific domain and scope of the proposed ontology. As stated in the previous sections, the initial objective was to support resource management in federated infrastructures for experimentation. The related phases to this management effort are depicted in Figure 54. Each step embodies a wide range of requirements and challenges. We particularly highlight the first phase in this section; however, our approach was to provide a hierarchical set of ontologies to cover the whole resource life-cycle.

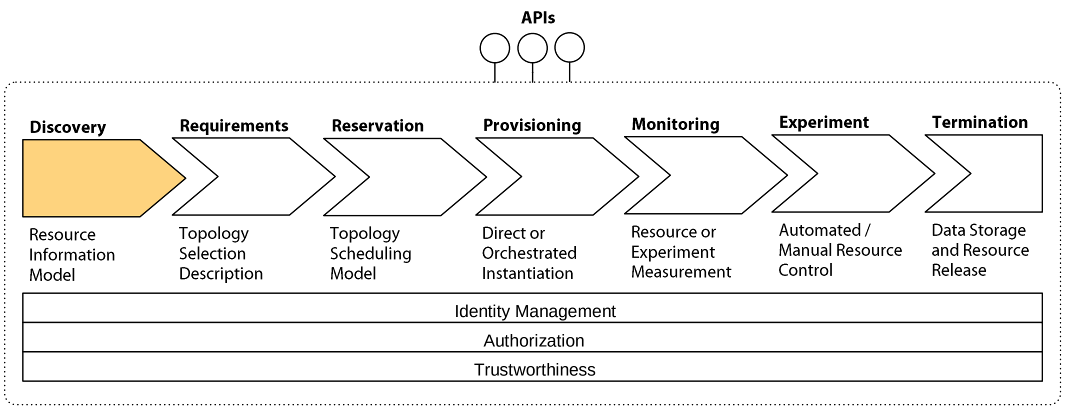

APIs
Discovery
Requirements
Reservation
Provisioning
Monitoring
Experiment
Termination
Resource Information Model
Topology Selection Description
Topology Scheduling Model
Direct or Orchestrated Instantiation
Resource or Experiment Measurement
Automated / Manual Resource Control
Data Storage and Resource Release
Identity Management
Authorization
Trustworthiness

Figure 54: The experiment life-cycle phases and protocols

10.3.1 Design

After identifying the scope of the reusable work (cf. Section 10.2), we have defined the significant concepts and properties. Consequently, our ontology bundle consists of nine ontologies, specifically the *omn* upper ontology and eight descendant ontologies (cf. Figure 55): *omn-federation*; *omn-lifecycle*; *omn-resource*; *omn-component*; *omn-service*; *omn-monitoring*; *omn-policy*; and domain-specific extensions called *omn-domain-xxx*.

Co-funded by the Horizon 2020 Framework Programme of the European Union

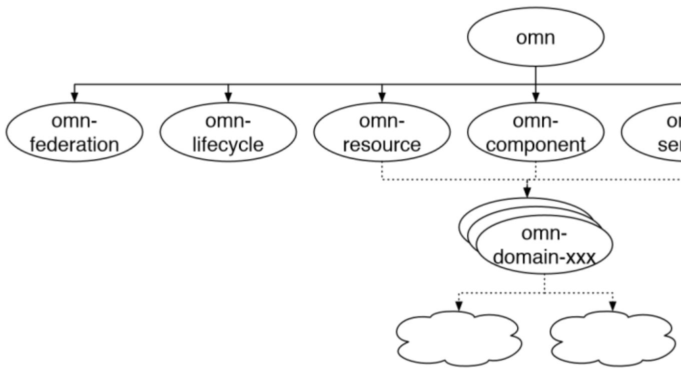


Figure 55: Open-Multinet ontology hierarchy

These ontologies can be used to describe formally a federation of e-infrastructures, including types and attributes of resources as well as services available within the federation. Ontologies also describe the life-cycle phases of usage. The various ontologies extend the upper OMN ontology (solid lines), which contains common concepts used within the other models. To describe concrete resources within a particular infrastructure, domain-specific ontologies might need to be defined that use and extend selected subsets of the OMN ontologies (dotted lines).

10.3.1.1 OMN Upper Ontology

The *omn* upper ontology defines the abstract terms required to describe federated infrastructures in general.

It includes a set of classes representing concepts providing general terms to model federated infrastructures, along with their respective components and services. These concepts are as follows:

- *Resource*: a stand-alone component of the infrastructure that can be provisioned, i.e., granted to a user such as a network node.
- *Service*: is a manageable entity that can be controlled and/or used via either APIs or capabilities that it supports, such as a SSH login.
- *Component*: constitutes a part of a *Resource* or a *Service*, such as a port of a network node.
- *Attribute*: helps to describe the characteristics and properties of a specific *Resource*, *Group*, *Resource*, or *Component*, such as Quality of Service (QoS).
- *Group*: is a collection of resources and services, for instance, a testbed or a requested network topology logically grouped together to perform a particular function.
- *Dependency*: describes a unidirectional relationship between two elements such as *Resource*, *Service*, *Component*, or *Group*. It may define, for example, an order in which particular resources need to be instantiated: first, a network link, and then, the compute nodes attached to it. This class opens up the possibility of adding more properties to a dependency via annotation.

Co-funded by the Horizon 2020 Framework Programme of the European Union

- *Layer*: describes a place within a hierarchy to which a specific *Group*, *Resource*, *Service*, or *Component* can adapt. Infrastructure resources naturally fall into layers, with resources at higher layers requiring presence of resources at lower layers in order to function.
- *Environment*: the conditions under which a *Resource*, *Group*, or *Service* is operating, as in, e.g., concurrent virtual machines.
- *Reservation*: a specification of a guarantee for a certain duration. Hence, it is a subclass of the "Interval" class of the W3C Time ontology [53].

The OMN upper ontology has 23 properties, of which the following are the most significant:

- *hasAttribute*: the Attribute associated with a *Component*, *Resource*, *Service*, or *Group*; e.g., CPU speed, or uptime.
- *hasComponent*: links a *Component* , *Resource*, or *Service* to its subcomponent.
- *hasGroup*: connects a *Group* to its subgroup; it is the inverse of *isGroupOf*.
- *hasReservation*: relates *Group*, *Resource* or *Service* to its *Reservation*.
- *hasResource*: declares that a specific *Group* has a *Resource*.
- *hasService*: declares that a *Group*, *Resource* or *Service* provides a *Service*.
- *withinEnvironment*: defines the "Environment" in which a *Group*, *Resource*, *Service*, or *Component* operates. An example of environment is the operating system under which a resource works.

To support rich querying and inferences, inverse counterparts have been declared for most properties. Figure 56 illustrates the key concepts and properties of the OMN ontology.

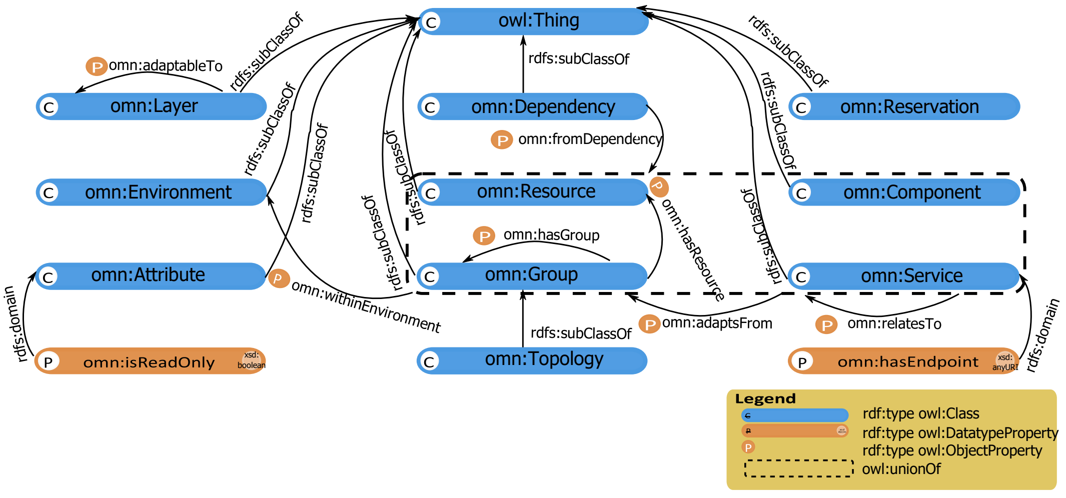


Figure 56: The key concepts and properties of the omn upper ontology

10.3.1.2 OMN Federation

A crucial part of the developed ontology set is the formal description of the relationship between the involved e-infrastructures (see Figure 57: Open-Multinet (OMN) Federation OntologyFigure 57). This allows to describe how resources relate to each other from the

Co-funded by the Horizon 2020 Framework Programme of the European Union

highest organizational level and depicts the starting point to discover capabilities and offered services. Federated providers maintain their autonomy in making resource-allocation decisions; however, they inter-operate with some federation-level functions, such as identity management or resource advertisement. To model these aspects, the *omn-federation* ontology introduces the concepts of a *Federation*, *FederationMember*, and *Infrastructure*, along with properties *hasFederationMember* and *isAdministeredBy*. The first two are subclasses of the schema:Organization class, which allows them to be described by properties of the Schema vocabulary. The latter concept relates infrastructures to a federation or to its members, and finally subclasses the *Group* concept which allows infrastructures to expose services with endpoints, such as an SFA Aggregate Manager (AM).

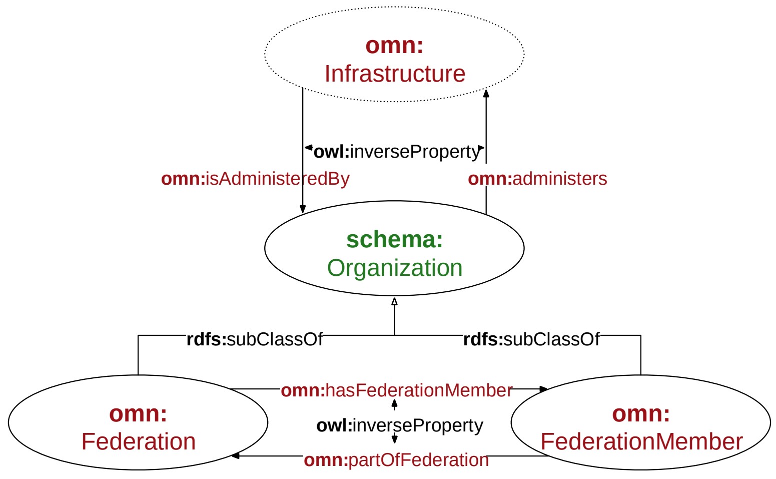
10.3.1.3 OMN Life Cycle

Another important ontology is the *omn-life cycle*, which addresses life-cycle management of a collection of resources and services (e.g., a requested network topology) that are grouped together to perform a particular function (e.g., to conduct an experiment or to deploy a service architecture). The life-cycle of the resources is described by a set of allocation and operational state changes such as

Allocated, *Provisioned*, *Unallocated*, *Pending*, *Ready*, *Started*, and *Stopped*. The life-cycle of the collection of resources reflects the first four phases of Figure 54:

1. the infrastructure provider advertises an *Offering* describing the available resources;
2. the user forms a *Request* defining the required collection of resources to the infrastructure provider;
3. the *Confirmation* contains an agreement by the provider, termed bound (tied to a specific set of physical resources) or unbound, to provide the requested resources;
4. and, finally, a *Manifest* describes the provisioned resources and their properties.

Each of these stages is represented as a subclass of the *Group* concept.



Co-funded by the Horizon 2020 Framework Programme of the European Union

Figure 57: Open-Multinet (OMN) Federation Ontology

10.3.1.4 OMN Monitoring

It describes the main concepts and relations to support monitoring services within federated infrastructures. It includes, therefore, multiple classes and properties that define measurement *Metrics*, their *Data*, *Unit*, *Tool*, and further *Generic-Concepts*. The monitoring ontology therefore comprises an upper-level ontology. It describes the common, basic concepts and properties, which then are reused and specialized in the subjacent ontologies.

10.3.1.5 OMN Resource

The OMN Resource ontology deals with the networking aspect of the infrastructure. It supports the creation of complex networks of interconnected resources. It include concepts and properties, e.g., *Node*, *Link*, and *IPAddress*, which are required for defining complex networks. It also supports defining single or bi-directional links, which can be utilized for defining the direction of packet flow across the link(s).

10.3.1.6 OMN Component

This ontology describes any entity that is part of a *Resource* or a *Service*; however, in itself, it is not a *Resource* or a *Service*. The OMN Component ontology describes concepts that are subclasses of the *Component* class defined in the OMN Upper ontology. It covers several classes to describe a set of basic entities in any Information and Communication Technology (ICT) infrastructure such as *CPU*, and *Memory*. Any class or instance of these can be the range of the property *hasComponent* that has a *Resource*, *Service*, or even another *Component* as a domain.

10.3.1.7 OMN Service

This ontology deals with ICT services. Any entity that delivers a value for its user is considered by the OMN Service ontology as a service. Examples can be services that offer APIs or login capabilities such as SSH. This ontology includes a set of classes to describe those services being used in ICT infrastructures. The current version covers a set of services being used and implemented by OMN ontology within the context of the application area addressed in this section, namely the FI experimentation.

10.3.1.8 OMN Policy

This ontology will cover policy-related concepts and relations. We consider the NOVI policy ontology as a starting point for its design, as it supports [10]:

- Authorization policies that specify authorization rights of users within the federation.
- Event-condition-action policies that enforce control and management actions upon certain events within the managed environment.
- Role-based-access control policies that assign users to roles, with different permissions/usage priorities on resources.
- Mission policies that define inter-platform obligations in a federation.

Co-funded by the Horizon 2020 Framework Programme of the European Union

10.3.1.9 OMN Domain Specific

OMN provides a way to define domain-specific ontologies, which customize the definition of concepts and relations for a particular ICT application. This allows a set of concepts and relations that are specific to a particular domain to be grouped along with some concepts and relations from other OMN ontologies to form a domain-specific ontology. Examples of these ontologies include, for instance, OMN Wireless ontology and OMN Cloud ontology, used to define the behavior of wireless networks and of cloud infrastructures, respectively. Another example includes the specification of an operating system (OS) version within a disk image, using *omn-domain-pc:hasDiskimageVersion*.

10.3.2 Use of Existing Ontologies

As described in 10.2 the OMN ontology set is inspired by and based on a number of existing formal information models. As an indicator, in Listing 1 a list of referenced vocabularies are shown that are used within the upper OMN ontology. An OMN Service, for example, has relationships to *novi:Service, dctype:Service, gr:ProductOrService, service:Service, schema:Service, nml:Service*, and *owl-s:Service*.

```
@prefix : <http://open−multinet.info/ontology/omn#> .
@prefix cc: <http://creativecommons.org/ns#> .
@prefix dc: <http://purl.org/dc/elements/1.1/> .
@prefix gr: <http://purl.org/goodrelations/v1#> .
@prefix nml: <http://schemas.ogf.org/nml/2013/05/base#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .

@prefix rdf: <http://www.w3.org/1999/02/22−rdf−syntax−ns#> .
@prefix xml: <http://www.w3.org/XML/1998/namespace#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix indl: <http://www.science.uva.nl/research/sne/indl#> .
@prefix move: <http://www.ontologydesignpatterns.org/cp/owl/move.owl#> .

@prefix novi: <http://fp7−novi.eu/im.owl#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf−schema#> .
@prefix time: <http://www.w3.org/2006/time#> .
@prefix vann: <http://purl.org/vocab/vann/> .
@prefix voaf: <http://purl.org/vocommons/voaf#> .

@prefix color: <http://geni−orca.renci.org/owl/app−color.owl#> .
@prefix owl−s: <http://www.daml.org/services/owl−s/1.0DL/Service.owl#> .
@prefix dctype: <http://purl.org/dc/dcmitype/> .
@prefix schema: <http://schema.org/> .
@prefix dcterms: <http://purl.org/dc/terms/> .
@prefix service: <http://purl.org/ontology/service#> .

@prefix collections: <http://geni−orca.renci.org/owl/collections.owl#> .
```

Listing 1: Used ontologies within omn.ttl

10.3.3 Implementation

We selected OWL2 to encode the OMN ontology suite due to its expressiveness, wide acceptance, and available tools. To ensure quality, changes to the ontologies are automatically checked using Apache Jena Eyeball inspectors; other validators such as the OntOlogy Pitfall Scanner (OOPS) [54] are executed manually.

Co-funded by the Horizon 2020 Framework Programme of the European Union

As part of the design process we are taking steps to ensure the broadest possible dissemination of the ontologies. As a result, we are using Dublin Core (DC), Vocabulary for Annotating Vocabulary Descriptions (VANN), and Vocabulary of a Friend (VOAF) vocabularies to describe the associated meta information. We are publishing the files by following best practices (http://www.w3.org/TR/swbpvocab-pub/). The URL http://open-multinet.info/ontology/omn provides both a human-readable documentation and machine-readable serializations. We have registered the permanent identifier https: //w3id.org/omn and published the root ontology to the Linked Open Vocabulary (LOV) repository

(http://lov.okfn.org/dataset/lov/vocabs/omn). Additionally, we have registered the *omn* name space (http://prefix.cc/omn). The source code of, and an issue tracker for, the ontologies are publicly available (https://github.com/w3c/omn).

In order to make the work recognizable to the international community, we established the Open-Multinet Forum, which is named after the ontology, and created the W3C OMN Community Group (https://www.w3.org/community/omn).

10.4 INFORMATION QUERYING AND VALIDATION

10.4.1 DBcloud Application For Federated Experimental Infrastructures

Most of the requirements for the development of OMN are rooted in research issues within the life-cycle management of resources across federated experimental infrastructures. In such a distributed environment, resource discovery is highly constrained, as it is based on (multi-) attribute matching. It requires an increased level of coordination between users and infrastructure providers as well as among infrastructure providers in the federation. For this purpose, we essentially propose a federation-wide knowledge layer over the federated infrastructures to support semantic representation of such information and to facilitate semantic-based resource discovery.

A large amount of semistructured information is available describing the GENI and FIRE testbed federations, including details about the testbeds involved and about the heterogeneous resources offered, reservation information, and monitoring data. This information is encoded mainly as human-readable text on websites as well as in the forms of JSON and XML trees via secured API calls. To extract this information and to make it semantically accessible on the Web, we previously introduced the OMN extraction framework [4].

In essence, the OMN extraction framework (Figure 58) follows the design of the DBpedia extraction framework [55]. Information is retrieved from the infrastructures, calling periodically according to methods of the SFA AM API (http://groups.geni.net/geni/wiki/GAPI_AM_API_V3_DETAILS). The downloaded documents are translated into a semantically annotated Resource Description Framework (RDF) [56] graph using the *OMN translator* and the OMN ontology suite. To extend the knowledge encoded in this graph, the Apache Jena inference engine is used within this process by applying infrastructure-specific rules.

Co-funded by the Horizon 2020 Framework Programme of the European Union

Finally, the resulting knowledge graph is written in an in-memory triplet database (Sesame v.2.8.6) and in a Turtle (TTL) [57] serialized file (DBcloud Dump). A SPARQL endpoint on top of the triplet data store implements a federation-wide lookup service that enables resource discovery by end-users. The result is currently available at http://lod.fed4fire.eu using, among others, the Vocabulary of Interlinked Datasets. The knowledge base currently describes approximately 100 aggregates, 3000 nodes, 30,000 links, and about 25,000 interfaces. This consists of 4.1 million statements, with the potential to grow substantially as new testbeds join the federation.

The *OMN translator* is a Java-based extensible translation mechanism introduced in [2], allowing the automated transformation of semi-structured data into an OMN based knowledge graph.

It translates statelessly between GENI, Resource Specifications (RSpecs), and OMN; applies inferencing rules for validation and knowledge injection; and has been extended to support Topology and Orchestration Specification for Cloud Applications (TOSCA) [58] and Yet Another Next Generation (YANG) [59] data models as well.

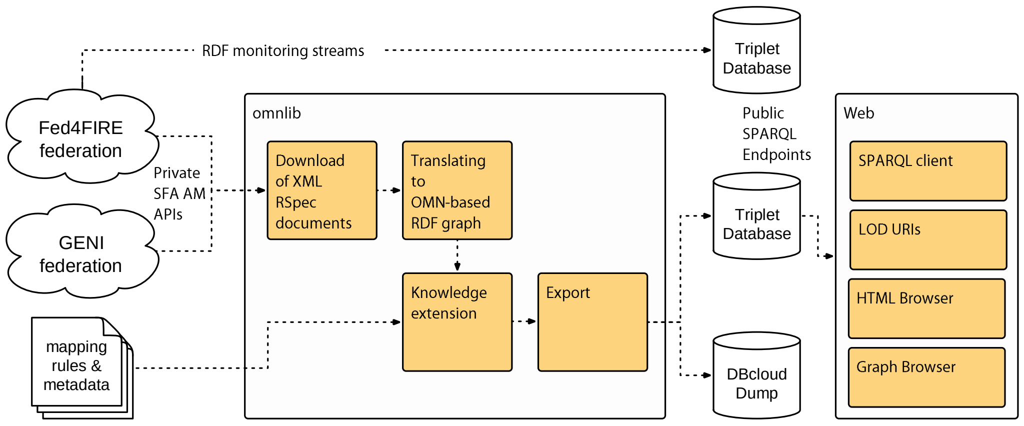


Figure 58: OMN extraction framework and lookup service (based on [4,55])

The implementation of the translation tool follows a Test Driven Development (TDD) approach, is included in a Continuous Integration (CI) environment with test coverage analytics, and is offered as a Java-based open-source library ("omnlib") in a public maven repository. It uses the Java Architecture for XML Binding (JAXB) and Apache Jena to map between XML, RDF, and Java objects. It supports a number of APIs: (i) a native API to be included in other Java projects; (ii) a CLI to be used within other applications; and (iii) a REST-based API to run as a Web service.

The *OMN translator* parses the XML tree and converts the tags and attributes to their corresponding classes or properties. To give a better understanding of this translation process, we provide an illustrative example for the conversion of a GENI Advertisement RSpec used to publish available resources within a federation of experimental infrastructures.

The example in Listing 2 shows a single node of type *PC* that can provision the sliver type

Co-funded by the Horizon 2020 Framework Programme of the European Union

PLAB-VSERVER (virtual server for PlanetLab). Traditionally, hardware type and sliver type used to be simple strings, but unique Uniform Resource Identifiers (URIs) are used here to provide machine-interpretable information.

```
<rspec xmlns="http://www.geni.net/resources/rspec/3" type="advertisement">
<node component_manager_id="urn:publicid:IDN+ple+authority+cm"
  component_id="urn:publicid:IDN+ple:netmodeple+node+stella.planetlab.ntua.gr"
exclusive="false" component_name="stella.planetlab.ntua.gr">                         Node Name
<hardware_type name="http://open−multinet.info/ontology/resources/pc#PC"/>               Hardware Type
<sliver_type name="http://open−multinet.info/ontology/resources/plab−vserver#PLAB−VSERVER"/>      Provisions Sliver Type
  <available now="true"/>
</node>
</rspec>
```

Listing 2: RSpec Advertisement (excerpt)

Listing 3 shows the converted graph, serialized in Turtle. The overall approach is to define an *omn:Topology* (the subclass *omn-lifecycle:Offering* is used in this case) that contains pointers to the offered resources. Each resource is an individual of a specific type that *can implement* (i.e., can provision) one or more specific sliver types.

```
<urn:uuid:7eb7b551−7566−4d3c−ac5f−f41a63236baa>   ← - - - - - - - - - - - - - - - - - - - Offering
        a <http://open−multinet.info/ontology/omn−lifecycle#Offering> ;
        <http://www.w3.org/2000/01/rdf−schema#label> "Offering" ;
        <http://open−multinet.info/ontology/omn#hasResource> <urn:publicid:IDN+ple:netmodeple+node+stella.planetlab.ntua.gr> .

<urn:publicid:IDN+ple:netmodeple+node+stella.planetlab.ntua.gr> a
        <http://open−multinet.info/ontology/omn−resource#Node> ,
          <http://open−multinet.info/ontology/resources/plab−vserver#PLAB−VSERVER> ,
           <http://open−multinet.info/ontology/resources/pc#PC> ;
        <http://www.w3.org/2000/01/rdf−schema#label>
                "stella.planetlab.ntua.gr"^^<http://www.w3.org/2001/XMLSchema#string> ;


        <http://open−multinet.info/ontology/omn#isResourceOf>
                <urn:uuid:7eb7b551−7566−4d3c−ac5f−f41a63236baa> ;

           <http://open−multinet.info/ontology/omn−lifecycle#canImplement>      ← - - - - - - - - - Implements Sliver Type
                <http://open−multinet.info/ontology/resources/plab−vserver#PLAB−VSERVER> ;
        <http://open−multinet.info/ontology/omn−lifecycle#hasComponentName> "stella.planetlab.ntua.gr" ;
        <http://open−multinet.info/ontology/omn−lifecycle#managedBy>
                <urn:publicid:IDN+ple+authority+cm> ;

        <http://open−multinet.info/ontology/omn−resource#hasHardwareType>      ← - - - -      Node      Hardware      Type
                <http://open−multinet.info/ontology/resources/pc#PC> ;
        <http://open−multinet.info/ontology/omn−resource#hasSliverType>
                <http://open−multinet.info/ontology/resources/plab−vserver#PLAB−VSERVER> ;
        <http://open−multinet.info/ontology/omn−resource#isAvailable> true ;
        <http://open−multinet.info/ontology/omn−resource#isExclusive> false .

<http://open−multinet.info/ontology/resources/plab−vserver#PLAB−VSERVER> a
        <http://open−multinet.info/ontology/omn−resource#SliverType> ;
        <http://open−multinet.info/ontology/omn−lifecycle#hasSliverName>
                "http://open−multinet.info/ontology/resources/plab−vserver#PLAB−VSERVER" .

<http://open−multinet.info/ontology/resources/pc#PC> a
        <http://open−multinet.info/ontology/omn−resource#HardwareType> ;
        <http://www.w3.org/2000/01/rdf−schema#label>
                "http://open−multinet.info/ontology/resources/pc#PC" .
```

Listing 3: OMN Offering

Co-funded by the Horizon 2020
Framework Programme of the European Union

10.4.2 Knowledge Extension and Information Querying

Having described the framework for semantic-based resource discovery in the context of federated experimental infrastructures, we will now focus on the specifics of the discovery process. Given a user request (query) and the aforementioned knowledge base, the resource-discovery problem amounts to automatically finding the resources from the triplet data store that match the query requirements along with policies set by infrastructure providers, since a request can be expressed at different levels of abstraction (*Resource Matching*). The adoption of the OMN ontology suite provides the necessary flexibility of expression as well as tools for querying and inference that simplify the typical problems encountered in the process of resource matching. Rules can capture domain background knowledge or infer resource requirements from the request model; specifically regarding the latter these are added as additional information to the initial request model. In addition, they can be used to check the request model's validity [49]. These benefits are highlighted in the following text.

10.4.2.1 Knowledge Extension

Background knowledge captures additional knowledge about the domain. This information can be used in matching a request with available resources. Knowledge is expressed in terms of rules that use the vocabulary of the ontology to add axioms. The knowledge graph can be extended by applying such rules.

For example, infrastructure providers in the federation do not advertise explicitly the hardware configurations of their resources in the RSpec XML documents provided. Such data are not translated into RDF. Instead, the information is encoded in each resource's *hardware type*, arbitrarily set by the infrastructure provider as highlighted in the advertisement excerpt provided in Listing 2, (i.e., *hardware type: PC*).

In Table 9, we provide sample hardware specifications for a subset of the federated experimental infrastructures as they are described by the corresponding infrastructure providers, namely, NETMODE (http://www.netmode.ntua.gr/testbed) and Virtual Wall 2 (http://doc.ilabt.iminds.be/ ilabt-documentation/virtualwallfacility.html) testbeds.

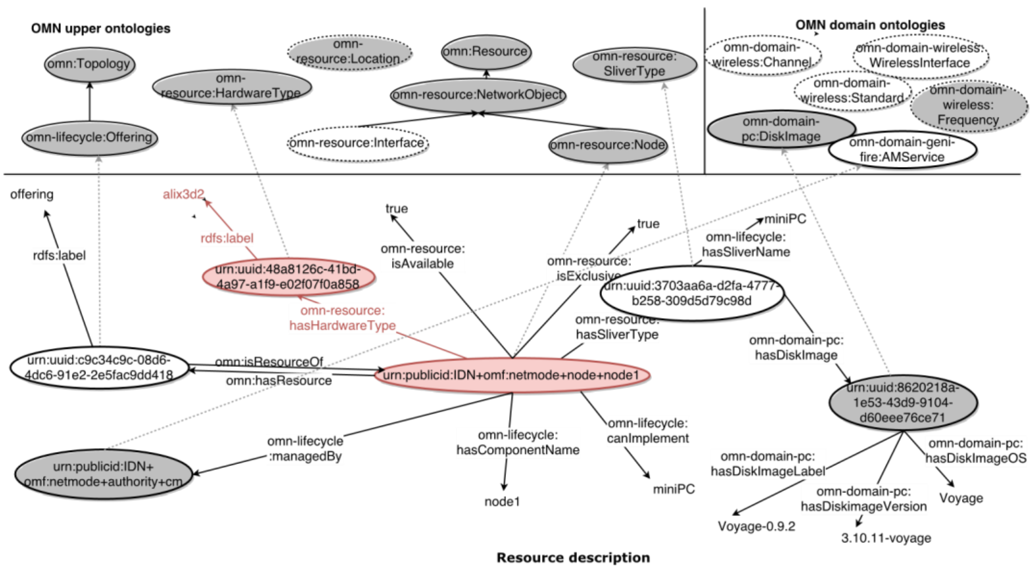
Table 9: Resource specifications

HW Type	Description
alix3d2	500 MHz AMD Geode LX800, 256 MB DDR DRAM, 1 GB flash card storage
pcgen3	2x Hexacore Intel E5645 (2.4 GHz) CPU, 24 GB RAM, 250 GB harddisk

Figure 59 depicts a rudimentary *offering* (advertisement) excerpt from the NETMODE infrastructure provide. For the sake of readability, only a single advertised resource is depicted (*omf.netmode.node1*). Moreover, the diagram does not show all the details of the resource description, although it identifies the distinct OMN ontologies used for this purpose, in the upper part the figure. In the excerpt provided the offered resource *omf.netmode.node1* is *managedBy* the infrastructure provider *omf:netmode* (*AMService*) and is part of (*isResourceOf*) the offering (advertisement) identified by *urn:uuid:c9c34c9c-08d6-4dc6-91e2-2e5fac9dd418*. The resource is related via the object property *hasHardwareType* to the *HardwareType* individual with the label *alix3d2*. It is associated (*hasSliverType*) to the *SliverType* individual, with the label *miniPC*,

Co-funded by the Horizon 2020 Framework Programme of the European Union

attributed with specific *Disk Image* properties (e.g., OS Voyage). As noted in this example, infrastructures advertise node capacities by their hardware type name (*alix3d2* in this case).



OMN upper ontologies
omn:Topology
omn-resource:HardwareType
omn-resource:Location
omn:Resource
omn-resource:SliverType
omn-resource:NetworkObject
omn-lifecycle:Offering
omn-resource:Interface
omn-resource:Node
OMN domain ontologies
omn-domain-wireless:Channel
omn-domain-wireless:WirelessInterface
omn-domain-wireless:Standard
omn-domain-wireless:Frequency
omn-domain-pc:DiskImage
omn-domain-geni-fire:AMService
offering
rdfs:label
alix3d2
rdfs:label
urn:uuid:48a8126c-41bd-4a97-a1f9-e02f07f0a858
omn-resource:hasHardwareType
true
omn-resource:isAvailable
omn-resource:isExclusive
true
miniPC
omn-lifecycle:hasSliverName
urn:uuid:3703aa6a-d2fa-4777-b258-309d5d79c98d
omn-resource:hasSliverType
omn-domain-pc:hasDiskImage
urn:uuid:c9c34c9c-08d6-4dc6-91e2-2e5fac9dd418
omn:isResourceOf
omn:hasResource
urn:publicid:IDN+omf:netmode+node+node1
urn:uuid:8620218a-1e53-43d9-9104-d60eee76ce71
omn-lifecycle:canImplement
omn-lifecycle:managedBy
omn-lifecycle:hasComponentName
urn:publicid:IDN+omf:netmode+authority+cm
node1
miniPC
omn-domain-pc:hasDiskImageLabel
omn-domain-pc:hasDiskImageOS
omn-domain-pc:hasDiskimageVersion
Voyage
Voyage-0.9.2
3.10.11-voyage
Resource description

Figure 59: Partial NETMODE offering

A simple example of background knowledge on the context of the “hardware type” is provided in Listing 4. The listing represents a subset of the rules used to expand the knowledge base with CPU-related information regarding *pcgen3* nodes listed in Table 9. Such information can be used in the resource matchmaking process. In the specific application, it is the responsibility of the federator, which maintains/provides the extraction framework, to apply such rules.

```
[rule1:
(?node omnres:hasHardwareType ?hwtype)
(?hwtype rdfs:label ?label) For every compute node with a hardware regex (?label , "pcgen0?3.*")
<- - - - - - - - - - - - - - - - - - - -
makeTemp(?cpuComp)                          type that has a label matching “pcgen0?3.*
–>
(?cpuComp rdf:type owl:NamedIndividual)
(?cpuComp rdf:type omncomp:CPU)
(?cpuComp rdfs:label "Intel E5645 CPU")        Insert standard information about this node
(?cpuComp omn:hasModelType "Intel E5645") <- - - - - - - - - -
(?cpuComp rdfs:label "Hexa Core Processor")    type: CPU Type, Core Count, CPU Frequency
(?cpuComp dbp:fastest "2.4"^^xsd:double)
(?cpuComp dbp:fastUnit <http://dbpedia.org/resource/GHZ>)
(?cpuComp omncomp:hasCores 6)

(?cpuComp dbp:arch <http://dbpedia.org/resource/X86−64>)
(?node omn:hasComponent ?cpuComp)       <- - - - Link new information to the compute node
]
```
Listing 4: Infrastructure knowledge 1 (excerpt)

Co-funded by the Horizon 2020 Framework Programme of the European Union

In our second example, shown in Listing 5, rules 1 to 3 mandate that each node identified by hardware type *alix3d2* have the hardware capacity described in Table 9 in terms of CPU, memory, and storage. Rules 4 to 6 link this information to the node.

```
[rule1: uriConcat(omncomp:,"alix3d2_mem", ?memComp) noValue(?memComp rdf:type owl:NamedIndividual)−> (?memComp
rdf:type owl:NamedIndividual)
(?memComp rdf:type omncomp:MemoryComponent)(?memComp omnmonunit:hasValue "256000000"^^xsd:integer) ]
[rule2: uriConcat(omncomp:,"alix3d2_cpu", ?cpuComp) noValue(?cpuComp rdf:type owl:NamedIndividual)−> (?cpuComp
rdf:type owl:NamedIndividual)
(?cpuComp rdf:type omncomp:CPU) (?cpuComp omnmonunit:hasValue "500000000"^^xsd:integer) ]
[rule3: uriConcat(omncomp:,"alix3d2_sto", ?stoComp) noValue(?stoComp rdf:type owl:NamedIndividual)−> (?stoComp
rdf:type owl:NamedIndividual)
(?stoComp rdf:type omncomp:StorageComponent)(?stoComp omnmonunit:hasValue "1000000000"^^xsd:integer) ]
[rule4: (?node omnres:hasHardwareType ?hwtype) (?hwtype rdfs:label "alix3d2"^^xsd:string) uriConcat(omncomp:,"alix3d2_mem",
      ?memComp) −> (?node omncomp:hasComponent ?memComp) ]
[rule5: (?node omnres:hasHardwareType ?hwtype) (?hwtype rdfs:label "alix3d2"^^xsd:string) uriConcat(omncomp:,"alix3d2_cpu",
   ?cpuComp) −> (?node omncomp:hasComponent ?cpuComp) ]
[rule6: (?node omnres:hasHardwareType ?hwtype) (?hwtype rdfs:label "alix3d2"^^xsd:string) uriConcat(omncomp:,"alix3d2_sto",
   ?stoComp) −> (?node omncomp:hasComponent ?stoComp) ]
```

Listing 5: Infrastructure knowledge 2 (excerpt)

4.2.2. Information Querying

Having applied the rules in Listing 4, a user may make a request for cloud resources with, for example, specific CPU requirements. In the sample SPARQL query provided in Listing 6, the user submits a request for two virtual machines with a specific number of CPU cores and OS type, e.g., Fedora:6cores. The results are shown in Listing 7.

```
SELECT ?resource1 ?resource2 WHERE {
?resource1 rdf:type omnres:Node .       <- - - - - - - - - - - - - - - - - - - Find me two hosts, resource1 and resource2
?resource2 rdf:type omnres:Node .
?resource1 omnres:hasSliverType/omndpc:hasDiskImage/omndpc:hasDiskimageOS ?os1.
?resource2 omnres:hasSliverType/omndpc:hasDiskImage/omndpc:hasDiskimageOS ?os2.
?resource1 omn:hasComponent ?cpuComp1.   <- - - - - - - - - - Both with 6 cores
    ?cpuComp1 rdf:type omncomp:CPU.
  ?cpuComp1 omncomp:hasCores ?cpuvalue1.FILTER (?cpuvalue1 = "6"^^xsd:integer).
?resource2 omn:hasComponent ?cpuComp2.
    ?cpuComp2 rdf:type omncomp:CPU.
  ?cpuComp2 omncomp:hasCores ?cpuvalue2.FILTER (?cpuvalue2 = "6"^^xsd:integer).

FILTER (xsd:string(?os1) = "Fedora"^^xsd:string).                 Both running Fedora
FILTER (xsd:string(?os2) = "Fedora"^^xsd:string).
FILTER (?resource1 = ?resource2)limit 1 ! !              Just one answer, please
```

Listing 6: SPARQL query 1

Listing 7: Query results

```
RESULTS
urn:publicid:IDN+wall2.ilabt.iminds.be+node+n095−05a
urn:publicid:IDN+wall2.ilabt.iminds.be+node+n096−02 TIME EXECUTION: 0.016sec
```

In a more complex example, a user may submit a request for *two nodes running a Linux distribution, with specific hardware requirements; e.g., 256MB of RAM and storage capacity greater than 500 MB*. The query is described in Listing 8. The resource-matching process is not straightforward, as it was in the previous case, even if we apply the rules in Listing 5. In most cases, Infrastructure Providers advertise the exact Linux Distribution (e.g., Voyage in Figure 6). Thus, the condition for *Linux OS variant* needs to be either incorporated into the request requirements or advertised explicitly by the testbeds. We follow the first approach in this case;

additional rules are added to infer automatically the resource characteristics, e.g., acceptable Linux distribution, without explicit statements needed from the user, as proposed in [60]. The rule set is an appropriately defined set of axioms from which additional implicit information can be derived. A sample rule used is provided in Listing 9 stating Linux compatibility (Voyage is a Linux-variant OS).

```
SELECT ?node1 ?node2 WHERE {
?node1 rdf:type omn_resource:Node.        <- - - - - - - - - - - - - - - - - - - Find me two hosts, node1 and node2

?node2 rdf:type omn_resource:Node.
?node1 omn:hasComponent ?memComp1.
?node2 omn:hasComponent ?memComp2.
?memComp1 rdf:type omn_component:MemoryComponent.
?memComp2 rdf:type omn_component:MemoryComponent.
?memComp1 omn_monitoring_unit:hasValue ?mvalue.
FILTER (?mvalue >= "256000000"^^xsd:integer)     <- - - - - - - - - Both with RAM greater than 256 MB
?memComp2 omn_monitoring_unit:hasValue ?mvalue.
FILTER (?mvalue >= "256000000"^^xsd:integer)
?node1 omn:hasComponent ?stoComp1.
?node2 omn:hasComponent ?stoComp2.
?stoComp1 rdf:type omn_component:StorageComponent.
?stoComp2 rdf:type omn_component:StorageComponent.
?stoComp1 omn_monitoring_unit:hasValue ?svalue1.
FILTER (?svalue1 >= "500000000"^^xsd:integer)    <- - - - - - - - - Both with disk storage greater than 500 MB
?stoComp2 omn_monitoring_unit:hasValue ?svalue2.
FILTER (?svalue2 >= "500000000"^^xsd:integer)


?node1  omn_resource:hasSliverType/omn_domain_pc:hasDiskImage/omn_domain_pc:hasDiskimageOS  ?os1.  <  Both  running  Linux
?node2      omn_resource:hasSliverType/omn_domain_pc:hasDiskImage/omn_domain_pc:hasDiskimageOS      ?os2.      FILTER
(xsd:string(?os1) = "Linux"^^xsd:string) FILTER (xsd:string(?os2) = "Linux"^^xsd:string)
FILTER (?node1 = ?node2)LIMIT 1
```

Listing 8: Initial SPARQL query 2

```
[rule7:(?node rdf:type omn−resource:Node)
(?node omn−resource:hasSliverType ?stype)
(?stype omn−domain−pc:hasDiskImage ?dimage)
(?dimage omn−domain−pc:hasDiskimageOS "Voyage"^^xsd:string) −>
(?dimage omn−domain−pc:hasDiskimageOS "Linux"^^xsd:string)]
```

Listing 9: IT background knowledge (excerpt)

Once the rules are applied, OR-AND clauses are built and added to the initial request [60]. Given the additional information injected into the graph, Listing 10 shows the new, expanded SPARQL query, with OR-AND clauses included in Lines 22–25. The results are restricted to one feasible matching solution, which is shown in Listing 11.

Listing 10: SPARQL query 2

Co-funded by the Horizon 2020
Framework Programme of the European Union

```
SELECT ?node1 ?node2 WHERE {
?node1 rdf:type omn_resource:Node.          <- - - - - - - - - - - - - - - - - - - Find me two hosts, node1 and node2

?node2 rdf:type omn_resource:Node.
?node1 omn:hasComponent ?memComp1.
?node2 omn:hasComponent ?memComp2.
?memComp1 rdf:type omn_component:MemoryComponent.
?memComp2 rdf:type omn_component:MemoryComponent.
?memComp1 omn_monitoring_unit:hasValue ?mvalue.
FILTER (?mvalue >= "256000000"^^xsd:integer)     <- - - - - - - - - - Both with RAM greater than 256 MB
?memComp2 omn_monitoring_unit:hasValue ?mvalue.
FILTER (?mvalue >= "256000000"^^xsd:integer)
?node1 omn:hasComponent ?stoComp1.
?node2 omn:hasComponent ?stoComp2.
?stoComp1 rdf:type omn_component:StorageComponent.
?stoComp2 rdf:type omn_component:StorageComponent.
?stoComp1 omn_monitoring_unit:hasValue ?svalue1.
FILTER (?svalue1 >= "500000000"^^xsd:integer)    <- - - - - - - - - - Both with disk storage greater than 500 MB
?stoComp2 omn_monitoring_unit:hasValue ?svalue2.
FILTER (?svalue2 >= "500000000"^^xsd:integer)                                                    Running Linux
?node1 omn_resource:hasSliverType/omn_domain_pc:hasDiskImage/omn_domain_pc:hasDiskimageOS ?os1.  <-
                                                                                             Variant
?node2 omn_resource:hasSliverType/omn_domain_pc:hasDiskImage/omn_domain_pc:hasDiskimageOS ?os2.
FILTER (xsd:string(?os1) = "Voyage"^^xsd:string ||xsd:string(?os1) = "Fedora"^^xsd:string || xsd:string(?os1) = "Ubuntu"
^^xsd:string || xsd:string(?os1) = "Linux"^^xsd:string)
FILTER (xsd:string(?os2) = "Voyage"^^xsd:string ||xsd:string(?os2) = "Fedora"^^xsd:string || xsd:string(?os2) = "Ubuntu" ^^xsd:string ||
xsd:string(?os2) = "Linux"^^xsd:string)
FILTER (?node1 = ?node2)LIMIT 1
```

```
RESULTS
:node1=> <urn:publicid:IDN+omf:netmode+node+node18>,
:node2=> <urn:publicid:IDN+omf:netmode+node+node14>
TIME EXECUTION: 0.299sec
```

Listing 11: Query results

10.4.3 Validation

Documents created using OMN vocabularies can be validated semantically in part by using traditional OWL entailments, which verify that the domains and ranges of properties used in a particular model match those defined in the vocabulary. We found, however, that the expressivity of those mechanisms was not always sufficient to validate the user requests being sent to the testbed. Procedural verification is not portable. It is hard to ensure correctness and consistency across implementations. To supplant traditional OWL mechanisms, we developed Datalog rule-sets that trigger inference errors when processing a document that either lacks specific information or is semantically ambiguous. In this section, we explore several examples of such rules.

For instance, if a user is attempting to request a network connection that loops to the same node on which it started, a request may be represented by a valid OMN model; however, semantically, it doesn’t make sense to the resource-matching algorithm that is attempting to reproduce the topology. To guard against cases like this, we validate the user’s request using the following Datalog rule in Listing 12.

```
(?Z rb:violation error('Connection Validation', 'Connection cannot loop on itself', ?Y))
  <− (?X rdf:type pc:PC), (?X nml:hasOutboundPort ?P1), (?X nml:hasInboundPort ?P2),
  (?Y rdf:type nml:Link), (?P1 nml:isSink ?Y), (?P2 nml:isSource ?Y) ]
```

Listing 12: Validating self-looping links in requests

Co-funded by the Horizon 2020 Framework Programme of the European Union

In some requests by end-users, every Virtual Machine (VM) node must specify an OS image to be booted. At the same time, a VM -server node does not need an image, since it operates using only a pre-determined image. The *pc* : *hasDiskImage* property is defined for all *PC* types, including VM Servers and VMs, so a cardinality restriction cannot be used in this case. This request validation rule is expressed as follows in Listing 13.

Listing 13: Validating presence of OS image in VM requests

```
(?Z rb:violation error("Validating that VM nodes have OS images", ?R)) <− (?R rdf:type pc:VM), noValue(?R,
pc:hasDiskImage, ?I)
```

It is important to emphasize that the set of the rules that we use continues to evolve with the schema and with the resource-matching algorithms used to allocate CI resources for the users. For example, as the algorithms become more sophisticated, they are able to function without some of the guards protecting them from poorly formed requests, reducing the need for some rules. Nonetheless, the designing of resource-matching and of embedding algorithms in testbeds is an active field of study. The availability of declarative rule-based semantic validation significantly simplifies the continuing evolution of these algorithms by clearly associating a particular algorithm with its own set of validation rules that prevent errant executions and simplify the algorithm code.

10.5 PERFORMANCE EVALUATION

By adopting formal information models and semantically annotated graphs, our approach allows operations to link, relate, enhance, query, and conduct logical manipulations of heterogeneous data, all of which would be impossible otherwise.

One of the most important measure for the applicability of our work is the amount of time required to translate and to query resources using our ontology. This time needs to range in a practicable span for the given context. Our initial work [4] looked at the sizes of the advertisements for testbeds in the FIRE and GENI projects and evaluated the performance of the translation to RDF of the respective

XML files. The novel work we present in this section show a more comprehensive comparison of the queries performance; namely, we look at the time needed to translate resource information to the one needed to list resources, as well as the performance of queries of different complexity.

We have analyzed the result of the *ListResources* method call of the 99 SFA AMs that are monitored (https://flsmonitor.fed4fire.eu/) within the Fed4FIRE project. This list contains 82 valid XML based GENI RSpec replies with 762.634 XML elements in total, of which 3.043 are *Nodes*, 31.155 are *Links*, and 25.493 *Interfaces*. Figure 60 shows the size of the RSpec advertisements in the testbeds we considered.

Co-funded by the Horizon 2020
Framework Programme of the European Union

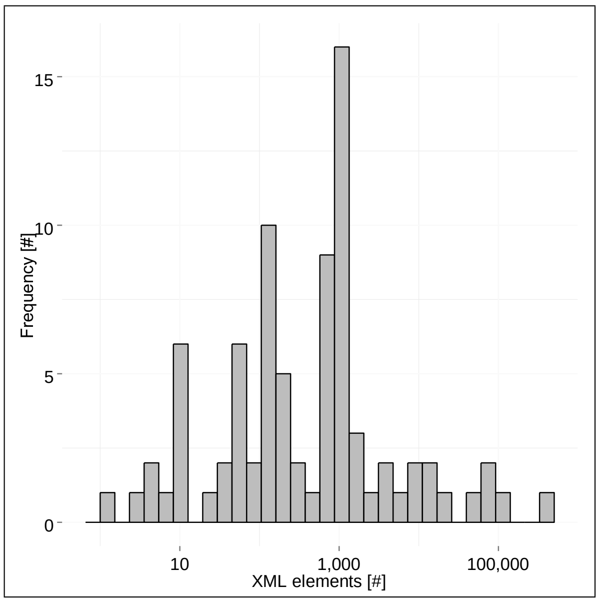


Figure 60: Size distribution of RSpec Advertisements (logarithmic)

To estimate the time needed to translate the advertisements, the actual RSpecs from these testbeds has been downloaded. The XML files were then translated to TTL serialized RDF graphs using the OMN translator. Of great importance to the potential scalability of our approach is the time taken for such translations, particularly with regard to the number of XML elements involved. 100 Advertisement RSpecs had been extracted, of which six contained errors, e.g., not adhering to the RSpec XML Schema Definition (XSD) file, and could not be translated without manual changes. Tests were run on a MacBookPro with OS X Yosemite, a 2.8 GHz Intel Core i7 processor, and 8 GB of RAM. Running a translation over all correct RSpecs produced median values of 24 milliseconds from XML to Java Architecture for XML Binding (JAXB) and 20 milliseconds from JAXB to RDF, yielding a total median translation time of 44 milliseconds from XML to RDF. As shown in Figure 61, translation times appear to be roughly linearly correlated with the number of XML elements translated, with a median of 180 elements and a maximum of 159,372 translated. This linear correlation indicates upwards scaling should be possible, although more data are required to confirm this point. At this stage, no major limiting factors have been identified, and, given appropriate processing power, translation should be possible in most foreseeable use cases.

Co-funded by the Horizon 2020 Framework Programme of the European Union

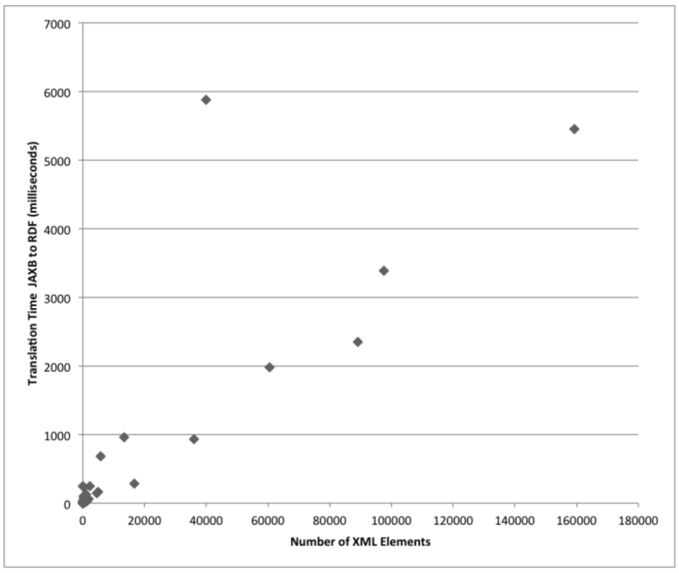


Figure 61: JAXB to RDF translation times versus number of XML elements [4]

To put the duration needed for the translation of an RSpec *Advertisement* into relation with the duration of the underlying function call needed in the FI experimentation context, we quantified the query and translation time for a single testbed. As indicated in Figure 60, about 95% of the testbeds expose fewer than 20.000 XML elements; therefore, we have used the CloudLab Wisconsin testbed (https://www.cloudlab.us), which exposes 19.371, for our measurements. The results in Figure 62 show that the average translation time of 583 ms ± 9 ms (95% CI) would add about 10% to the average response time of 5453 ms ± 131 ms (95% CI). This effect, however, could be mitigated by translating in advance or by distributing the work load. The delay of over 5 seconds for listing resources using a single API call, is influenced by mainly two factors. First, the available bandwidth to transmit the resulting XML document from the testbed to the caller. Second, the testbed internal communication architecture to gather the required information, as CloudLab is a distributed infrastructure itself that is composed by three different sites.

Co-funded by the Horizon 2020 Framework Programme of the European Union

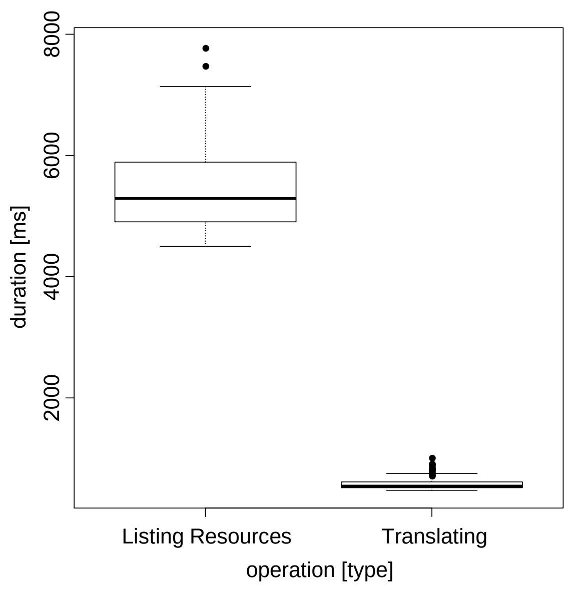


Figure 62: Listing/translating resource information

Assuming that a testbed accepts the potentially enhanced response time, in favor of the added value of merging its information into a global linked data set, its resources can be found by applying the aforementioned resource-matching queries. The translation of all available tree data structures into an RDF-based graph, using our OMN vocabulary and rules, resulted in a set of 2.911.372 statements. It builds the basis for our conclusion that adding further rules, infrastructures, and other data sources will increase the potential for significant growth.

In Figure 63, the duration of Listing 10 against this graph is shown. To assess the performance impact of the complexity of the query, it has been compared with a simpler one, which is shown in Listing 14 together with its result in Listing 15. While finding the three largest aggregates took on average 129 ms ± 3 ms (95% CI), the matching query took on average 168 ms ± 1 ms (95% CI) and therefore took about 30% longer, yet much less time than a single *ListResources* call in a single testbed. Finally, we have summarized our findings in Table 10.

Co-funded by the Horizon 2020
Framework Programme of the European Union

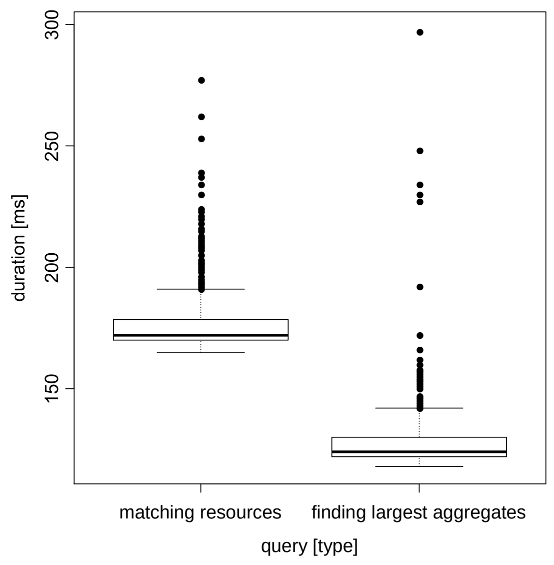


Figure 63: Performance comparison of queries

Listing 14: Finding the largest aggregate via query

```
SELECT (COUNT(?am) as ?fre) ?am WHERE {
   ?node omn−lifecycle:managedBy ?am .
} GROUP BY (?am) ORDER BY DESC (?fre) LIMIT 3
```

```
?fre ?am
719 <urn:publicid:IDN+emulab.net+authority+cm>
326 <urn:publicid:IDN+utah.cloudlab.us+authority+cm>
255 <urn:publicid:IDN+ple+authority+cm>
```

Listing 15: Largest aggregates

Table 10: Results of the performance evaluation

Median Duration [ms]	Phase
24	Translation from XML to JAXB (on average)
20	Translation from JAXB to RDF (on average)
583	Translation of 19.371 XML elements (CloudLab)
5453	Listing resources (CloudLab)
129	Querying three largest aggregates (Listing 10))
168	Matching resources (Listing 14)

Co-funded by the Horizon 2020 Framework Programme of the European Union

10.6 CONCLUSION AND FUTURE WORK

The OMN set of ontologies that we presented in this article has been developed to support resource management in federated and distributed computing infrastructures. OMN provides a federation-wide knowledge layer that eases the process of resource selection and matching.

In this article, we described the OMN framework, which allows the extraction of underlying information from tree-based data structures. It exposes this information in the form of OMN triples to interested parties via the Web. DBcloud is an application developed in support of the federation of experimental cyber-infrastructures which relies on OMN translators that automatically transform semi-structured data into OMN graphs. An important aspect that we have assessed is the performance of such translations, as this is crucial to OMN usability and adoption. We have shown that the translation and query times require additional time (on the order of 10% in our experiment), which, however, we expect to be acceptable to all resource providers given the added value of merging information.

We have also shown how users can query OMN information that represents the resources available in the underlying infrastructures and match them with their own computational requirements. In such case, we evaluated the time needed to find matching resources. We have shown that more complex queries complete within times that are acceptable to end-users.

In the long run, we expect that our contributions will outlive the specific use case of the cloud testbed resource management. We believe that it will be accepted by the broader community of academic and commercial cloud providers. It will help to create an ecosystem of flexible, extensible tools and mechanisms that will see the use of cloud platforms become even more pervasive. We expect it to open up the marketplace to competing cloud providers, large and small, catering to specific market niches. We are also promoting adoption of OMN in new domains such as the Internet of Things (IoT). As a specific Industrial Internet of Things (IIoT) [61] example, things, services and data can be connected between federated manufacturing facilities. As an analog to the federation of testbeds the involved facilities, the digital factories, their available APIs and services, have to be described formally to allow for matchmaking capabilities required for the envisioned autonomous production within the fourth industrial revolution. Our ontology set could act as a basis. Following discussions within the German initiative Plattform Industrie 4.0 (PI4.0), linking information based on the LOD paradigm and using interfaces such as the W3C WoT could build a technological base for implementing this vision. Another focus area for the OMN ontologies is the integration with ontologies defining data-access policies among cooperating entities that make use of the cloud infrastructures. The support provided by OMN for the definition of complex usage of heterogeneous resources will be the backbone for novel kinds of open data services, both in industrial and commercial settings, as well as in the scientific community.

10.7 ABBREVIATIONS

The following abbreviations are used in this manuscript:

Co-funded by the Horizon 2020
Framework Programme of the European Union

AC4	Activity Chain 4—Service Openness and Interoperability Issues/Semantic Interoperability
AM	Aggregate Manager
API	Application Programming Interface
DC	Dublin Core
ETSI	European Telecommunications Standards Institute
FCFA	Federated Cloud Framework Architecture
Fed4FIRE	Federation for FIRE
FI	Future Internet
FIRE	Future Internet Research and Experimentation
GENI	Global Environment for Network Innovations
GLUE	Grid Laboratory for a Uniform Environment
GR	Good Relations
IaaS	Infrastructure as a Service
ICT	Information and Communication Technology
IEEE	Institute of Electrical and Electronics Engineers
IERC	European Research Cluster on the Internet of Things
IIoT	Industrial Internet of Things
IMF	Information Modeling Framework
INDL	Infrastructure and Network Description Language
IoT	Internet of Things
JAXB	Java Architecture for XML Binding
JSON	JavaScript Object Notation
LDAP	Lightweight Directory Access Protocol
LOD	Linked Open Data
M2M	Machine-To-Machine Communication
MAS	OneM2MWorking Group 5 Management, Abstraction and Semantics
mOSAIC	Open-Source API and Platform for Multiple Clouds
NDL-OWL	Network Description Language based on the Web Ontology Language
NML	Network Mark-Up Language
NOVI	Networking innovations Over Virtualized Infrastructures
OGF	Open Grid Forum
OMN	Open-Multinet
OOPS	OntOlogy Pitfall Scanner
OWL-S	Semantic Markup forWeb Services
P2302	Standard for Intercloud Interoperability and Federation
PI4.0	Plattform Industrie 4.0
QoS	Quality of Service
RDF	Resource Description Framework
RSpec	Resource Specification
S-OGSA	Semantic Open Grid Service Architecture
SFA	Slice-based Federation Architecture
SPARQL	SPARQL Protocol And RDF Query Language
SQL	Structured Query Language
SSH	Secure Shell
SSN	Semantic Sensor Network
TOSCA	Topology and Orchestration Specification for Cloud Applications
TTL	Turtle
UCI	Unified Cloud Interface
URL	Uniform Resource Locator
VANN	Vocabulary for Annotating Vocabulary Descriptions

Co-funded by the Horizon 2020 Framework Programme of the European Union

VOAF	Vocabulary of a Friend
VoID	Vocabulary of Interlinked Datasets
W3C	WorldWide Web Consortium
WoT	Web of Things
XML	Extensible Markup Language
XSD	XML Schema Definition
YANG	Yet Another Next Generation

10.8 REFERENCES

1. Ashton, K. That 'Internet of Things' Thing. *RFID J.* **2009**, *6*, 4986.
2. Willner, A.; Papagianni, C.; Giatili, M.; Grosso, P.; Morsey, M.; Al-Hazmi, Y.; Baldin, I. The open-multinet upper ontology towards the semantic-based management of federated infrastructures. In Proceedings of the 10th EAI International Conference on Testbeds and Research Infrastructures for the Development of Networks & Communities, Vancouver, BC, Canada, 24–25 June 2015; p. 10.
3. Berners-Lee, T.; Hendler, J.; Lassila, O. The Semantic Web. *Sci. Am.* **2001**, *284*, 34–43.
4. Morsey, M.; Willner, A.; Loughnane, R.; Giatili, M.; Papagianni, C.; Baldin, I.; Grosso, P.; Al-Hazmi, Y. DBcloud: Semantic Dataset for the cloud. In Proceedings of the 2016 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), San Francisco, CA, USA, 10–15 April 2016; pp. 207–212.
5. Berman, M.; Chase, J.S.; Landweber, L.; Nakao, A.; Ott, M.; Raychaudhuri, D.; Ricci, R.; Seskar, I. GENI: A federated testbed for innovative network experiments. *Comput. Netw.* **2014**, *61*, 5–23.
6. Gavras, A.; Karila, A.; Fdida, S.; May, M.; Potts, M. Future internet research and experimentation. *ACM SIGCOMM Comput. Commun. Rev.* **2007**, *37*, 89–92.
7. Bauer, F.; Kaltenböck, M. *Linked Open Data: The Essentials*; Edition Mono/Monochrom: Vienna, Austria, 2011.
8. Ghijsen, M.; van der Ham, J.; Grosso, P.; de Laat, C. Towards an infrastructure description language for modeling computing infrastructures. In Proceedings of the 10th IEEE International Symposium on Parallel and Distributed Processing with Applications, Madrid, Spain, 10–13 July 2012; pp. 207–214.
9. Ghijsen, M.; van der Ham, J.; Grosso, P.; Dumitru, C.; Zhu, H.; Zhao, Z.; de Laat, C. A Semantic-Web Approach for Modeling Computing Infrastructures. *Comput. Electr. Eng.* **2013**, *39*, 2553–2565.
10. van der Ham, J.; Stéger, J.; Laki, S.; Kryftis, Y.; Maglaris, V.; de Laat, C. The NOVI information models. *Future Gener. Comput. Syst.* **2015**, *42*, 64–73.
11. Andreozzi, S.; Burke, S.; Field, L.; Fisher, S.; Konya, B.; Mambelli, M.; Schopf, J.M.; Viljoen, M.; Wilson, A. *GFD 147: Glue Schema Specification*; Open Grid Forum (OGF): Muncie, IN, USA, 2007.
12. Drozdowicz, M.; Ganzha, M.; Paprzycki, M.; Olejnik, R.; Lirkov, I.; Telegin, P.; Senobari, M. Ontologies, agents and the grid: An overview. In *Parallel, Distributed and Grid Computing for Engineering*; Saxe-Coburg Publications: Stirlingshire, UK, 2009; pp. 117–140.
13. Corcho, O.; Alper, P.; Kotsiopoulos, I.; Missier, P.; Bechhofer, S.; Goble, C. An overview of S-OGSA: A reference semantic grid architecture. *Web Semant. Sci. Serv. Agents World Wide Web* **2006**, *4*, 102–115.
14. Junghans, M.; Agarwal, S.; Studer, R. Towards practical semantic web service discovery. In Proceedings of the European Semantic Web Conference, Heraklion, Greece, 30 May–3 June 2010; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2010; pp. 15–29.
15. Stollberg, M.; Keller, U.; Lausen, H.; Heymans, S. In *Two-Phase Web Service Discovery Based on Rich Functional Descriptions*, *Proceedings of the 4th European Semantic Web Conference, Innsbruck, Austria, 3–7 June 2007*; Franconi, E., Kifer, M., May, W., Eds.; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2007; pp. 99–113.

Co-funded by the Horizon 2020
Framework Programme of the European Union

16. Paolucci, M.; Kawamura, T.; Payne, T.R.; Sycara, K. In *Semantic Matching of Web Services Capabilities, Proceedings of the International Semantic Web Conference, Sardinia, Italy, 9–12 June 2002*; Springer: Berlin/Heidelberg, Germany, 2002; pp. 333–347.
17. Pedrinaci, C.; Cardoso, J.; Leidig, T. Linked USDL: A vocabulary for web-scale service trading. In *The Semantic Web: Trends and Challenges*; Springer: Cham, Switzerland, 2014; pp. 68–82.
18. Cardoso, J.; Barros, A.P.; May, N.; Kylau, U. Towards a Unified Service Description Language for the Internet of Services: Requirements and First Developments. In Proceedings of the 2010 IEEE International Conference on Services Computing (SCC), Miami, FL, USA, 5–10 July 2010; pp. 602–609.
19. Oberle, D.; Barros, A.P.; Kylau, U.; Heinzl, S. A unified description language for human to automated services. *Inf. Syst.* **2013**, *38*, 155–181.
20. Martin, D.; Burstein, M.; Hobbs, J.; Lassila, O.; McDermott, D.; McIlraith, S.; Narayanan, S.; Paolucci, M.; Parsia, B.; Payne, T.; et al. OWL-S: Semantic Markup for Web Services. *Word Wide Web Consortium (W3C)*. Available online: https://www.w3.org/Submission/OWL-S/ (accessed on 20 June 2017).
21. Hepp, M. GoodRelations: An ontology for describing products and services offers on the web. In *Knowledge Engineering: Practice and Patterns*; Springer: Berlin/Heidelberg, Germany, 2008; pp. 329–346.
22. Youseff, L.; Butrico, M.; Da Silva, D. Toward a unified ontology of cloud computing. In Proceedings of the IEEE Grid Computing Environments Workshop, Austin, TX, USA, 16 November 2008; pp. 1–10.
23. Han, T.; Sim, K.M. In *An ontology-enhanced cloud service discovery system*. *Proceedings of the International MultiConference of Engineers and Computer Scientists*, Hong Kong, China, 17–19 March 2010; Springer: Berlin/Heidelberg, Germany, 2010; Volume 1, pp. 17–19.
24. Ma, Y.B.; Jang, S.H.; Lee, J.S. Ontology-Based Resource Management for Cloud Computing. In *Intelligent Information and Database Systems*; Springer: Berlin/Heidelberg, Germany, 2011; pp. 343–352.
25. Haase, P.; Mathäß, T.; Schmidt, M.; Eberhart, A.; Walther, U. Semantic technologies for enterprise cloud management. In Proceedings of the International Semantic Web Conference, Shanghai, China, 7–11 November 2010; pp. 98–113.
26. Haak, S.; Grimm, S. Towards custom cloud services—Using semantic technology to optimize resource configuration. In Proceedings of the 8th Extended Semantic Web Conference (ESWC 2011), Paris, France, 20–25 October 2011; pp. 345–359.
27. Grozev, N.; Buyya, R. Inter-Cloud architectures and application brokering: Taxonomy and survey. *Softw. Pract. Exp.* **2014**, *44*, 369–390.
28. Manno, G.; Smari, W.W.; Spalazzi, L. FCFA: A semantic-based federated cloud framework architecture. In Proceedings of the International Conference on High Performance Computing & Simulation (HPCS), Madrid, Spain, 02–06 July 2012; pp. 42–52.
29. Bernstein, D.; Deepak, V.; Chang, R. *Draft Standard for Intercloud Interoperability and Federation (SIIF)*; Technical Report IEEE P2303; IEEE Computer Society: Washington, DC, USA, 2015.
30. Martino, B.D.; Cretella, G.; Esposito, A.; Willner, A.; Alloush, A.; Bernstein, D.; Vij, D.; Weinman, J. Towards an ontology-based intercloud resource catalogue—The IEEE P2302 intercloud approach for a semantic resource exchange. In Proceedings of the International Conference on Cloud Engineering, Tempe, AZ, USA, 9–13 March 2015; pp. 458–464.
31. Moscato, F.; Aversa, R.; Di Martino, B.; Fortis, T.; Munteanu, V. An analysis of mOSAIC ontology for cloud resources annotation. In Proceedings of the Federated Conference on Computer Science and Information Systems (FedCSIS), Szczecin, Poland, 18–21 September 2011; pp. 973–980.
32. Santana-Pérez, I.; Perez-Hernández, M.S. A semantic scheduler architecture for federated hybrid clouds. In Proceedings of the 2012 IEEE 5th International Conference on Cloud Computing (CLOUD), Honolulu, HI, USA, 24–29 June 2012; pp. 384–391.
33. Dastjerdi, A.V.; Tabatabaei, S.G.H.; Buyya, R. An effective architecture for automated appliance management system applying ontology-based cloud discovery. In Proceedings of the 10th IEEE/ACM International

Conference on Cluster, Cloud and Grid Computing (CCGrid), Washington, DC, USA, 17–20 May 2010; pp. 104–112.
34. Ngan, L.D.; Kanagasabai, R. OWL-S based semantic cloud service broker. In Proceedings of the 2012 IEEE 19th International Conference on Web Services, Honolulu, HI, USA, 24–29 June 2012; pp. 560–567.
35. Aranda, C.B.; Corby, O.; Das, S.; Feigenbaum, L.; Gearon, P.; Glimm, B.; Harris, S.; Hawke, S.; Herman, I.; Humfrey, N.; et al. SPARQL 1.1 Overview. *Word Wide Web Consortium (W3C)*. Available online: https://www.w3.org/TR/2012/PR-sparql11-overview-20121108/ (accessed on 20 June 2017).
36. Serrano, M.; Barnaghi, P.; Cousin, P. *IoT Semantic Interoperability: Research Challenges, Best Practices, Solutions and Next Steps*; Technical Report for European Research Cluster on the Internet of Things (IERC): Brussels, Belgium, 2013.
37. Compton, M.; Barnaghi, P.; Bermudez, L.; García-Castro, R.; Corcho, O.; Cox, S.; Graybeal, J.; Hauswirth, M.; Henson, C.; Herzog, A.; et al. The SSN ontology of the W3C semantic sensor network incubator group. *Web Semant. Sci. Serv. Agents World Wide Web* **2012**, *17*, 25–32.
38. Wu, G..; Talwar, S.; Johnsson, K.; Himayat, N.; Johnson, K.D. M2M: From mobile to embedded internet. *IEEE Commun. Mag.* **2011**, *49*, 36–43.
39. Cacˇkovic´, V.; Popovic´, Ž. Abstraction and Semantics support in M2M communications. In Proceedingsˇ of the 36th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), Opatija, Croatia, 20–24 May 2013; pp. 404–408.
40. European Telecommunications Standards Institute. *Machine-to-Machine Communications (M2M); Functional Architecture*; Technical Report for ETSI: Valbonne, France, 2013.
41. Swetina, J.; Lu, G.; Jacobs, P.; Ennesser, F.; Song, J. Toward a standardized common M2M service layer platform: Introduction to oneM2M. *IEEE Wirel. Commun.* **2014**, *21*, 20–26.
42. Raggett, D. Web of Things Framework. *Word Wide Web Consortium (W3C)*. Available online: https://www.w3.org/2015/05/wot-framework.pdf (accessed on 20 June 2017).
43. Vandenberghe, W.; Vermeulen, B.; Demeester, P.; Willner, A.; Papavassiliou, S.; Gavras, A.; Quereilhac, A.; Al-Hazmi, Y.; Lobillo, F.; Velayos, C.; et al. Architecture for the heterogeneous federation of future internet experimentation facilities. In Proceedings of the Future Network and Mobile Summit (FNMS), Lisboa, Portugal, 3–5 July 2013; pp. 1–11.
44. Peterson, L.; Sevinc, S.; Lepreau, J.; Ricci, R. *Slice-Based Federation Architecture*, version 2; GENI: West Hollywood, CA, USA, 2010.
45. Van der Ham, J.; Dijkstra, F.; Lapacz, R.; Zurawski, J. *GFD 206: Network Markup Language Base Schema*; Technical Report for Open Grid Forum (OGF): Muncie, IN, USA, 2013.
46. Escalona, E.; Peng, S.; Nejabati, R.; Simeonidou, D.; Garcia-Espin, J.A.; Riera, J.F.; Figuerola, S.; de Laat, C.
GEYSERS: A novel architecture for virtualization and co-provisioning of dynamic optical networks and IT services. In Proceedings of the Future Network and Mobile Summit, Warsaw, Poland, 15–17 June 2011; pp. 1–8.
47. Garcia-Espin, J.A.; Riera, J.F.; Figuerola, S.; Ghijsen, M.; Demchenko, Y.; Buysse, J.; de Leenheer, M.; Develder, C.; Anhalt, F.; Soudan, S. Logical infrastructure composition layer, the GEYSERS holistic approach for infrastructure virtualisation. In Proceedings of the Terena Networking Conference (TNC), Reykjavík, Iceland, 21–24 May 2012; pp. 1–16.
48. Baldine, I.; Xin, Y.; Mandal, A.; Renci, C.H.; Chase, U.C.J.; Marupadi, V.; Yumerefendi, A.; Irwin, D. Networked cloud orchestration: A GENI perspective. In Proceedings of the Globecom Workshops, Miami, FL, USA, 6–10 December, 2010; pp. 573–578.
49. Yufeng, X.; Baldine, I.; Chase, J.; Anyanwu, K. *TR-13-02: Using Semantic Web Description Techniques for Managing Resources in a Multi-Domain Infrastructure-as-a-Service Environment*; Technical Report for RENCI: Durham, NC, USA, 2013.

Co-funded by the Horizon 2020
Framework Programme of the European Union

50. Xin, Y.; Hill, C.; Baldine, I.; Mandal, A.; Heermann, C.; Chase, J. *Semantic Plane: Life Cycle of Resource Representation and Reservations in a Network Operating System*; Technical Report for RENCI: Durham, NC, USA, 2013.
51. Xin, Y.; Baldin, I.; Chase, J.; Ogan, K.; Anyanwu, K. *Leveraging Semantic Web Technologies for Managing Resources in a Multi-Domain Infrastructure-as-a-Service Environment*; Technical Repor for RENCI: Durham, NC, USA, 2014.
52. Noy, N.F.; Mcguinness, D.L. *Ontology Development 101: A Guide to Creating Your First Ontology*; Technical Report for Stanford University: Stanford, CA, USA, 2001.
53. Hobbs, J.R.; Pan, F. Time Ontology in OWL. *Word Wide Web Consortium (W3C)*. Available online: https://www.w3.org/TR/owl-time/ (accessed on 20 June 2017).
54. Poveda-Villalón, M.; Suárez-Figueroa, M.C.; Gómez-Pérez, A. Validating ontologies with OOPS! In *Knowledge Engineering and Knowledge Management*; Springer: Berlin/Heidelberg, Germany, 2012; pp. 267–281.
55. Lehmann, J.; Isele, R.; Jakob, M.; Jentzsch, A.; Kontokostas, D.; Mendes, P.N.; Hellmann, S.; Morsey, M.; van Kleef, P.; Auer, S.; et al. DBpedia-a large-scale, multilingual knowledge base extracted from wikipedia. *Semant. Web J. SWJ* **2014**, *5*, 1–29.
56. Cyganiak, R.; Wood, D.; Lanthaler, M. Resource Description Framework (RDF) 1.1 Concepts and Abstract Syntax. *Word Wide Web Consortium (W3C)*. Available online: http://travesia.mcu.es/portalnb/jspui/handle/10421/2427 (accessed on 20 June 2017).
57. Beckett, D.; Berners-Lee, T.; Prud'hommeaux., E. Turtle-Terse RDF Triple Language. *Word Wide Web Consortium (W3C)*. Available online: https://www.w3.org/TeamSubmission/turtle/ (accessed on 20 June 2017).
58. OASIS. *Topology and Orchestration Specification for Cloud Applications (TOSCA)*, version 1; OASIS Standard; Advancing Open Standards for the Information Society (OASIS): Burlington, MA, USA, 2013.
59. Bjorklund, M. *RFC 6020: YANG—A Data Modeling Language for the Network Configuration Protocol (NETCONF)*; RFC 6020 (Proposed Standard); RFC Editor: Los Angeles, CA, USA, 2010.
60. Urbani, J.; van Harmelen, F.; Schlobach, S.; Bal, H. QueryPIE: Backward reasoning for OWL Horst over very large knowledge bases. In Proceedings of the 10th International Semantic Web Conference (ISWC'11), Berlin/Heidelberg, Germany, 23–27 October 2011; pp. 730–745.
61. Jeschke, S.; Brecher, C.; Song, H.B.; Rawat, D.B. *Industrial Internet of Things*; Springer: Cham, Switzerland, 2017.

10.9 DESCRIPTION OF MATCHMAKING CAPABILITIES

For the first prototype, a hybrid version of a Semantic Web search was chosen. This approach is presented in the paper "A Hybrid Approach for Searching in the Semantic Web".

The idea is that the user starts a search with keywords, whereupon the matchmaking system searches for these terms in the knowledge graph, first using a traditional search engine, and then expands the list of results with a technique called "Spread Activation". Figure 64 shows the architecture of this approach.

Co-funded by the Horizon 2020
Framework Programme of the European Union

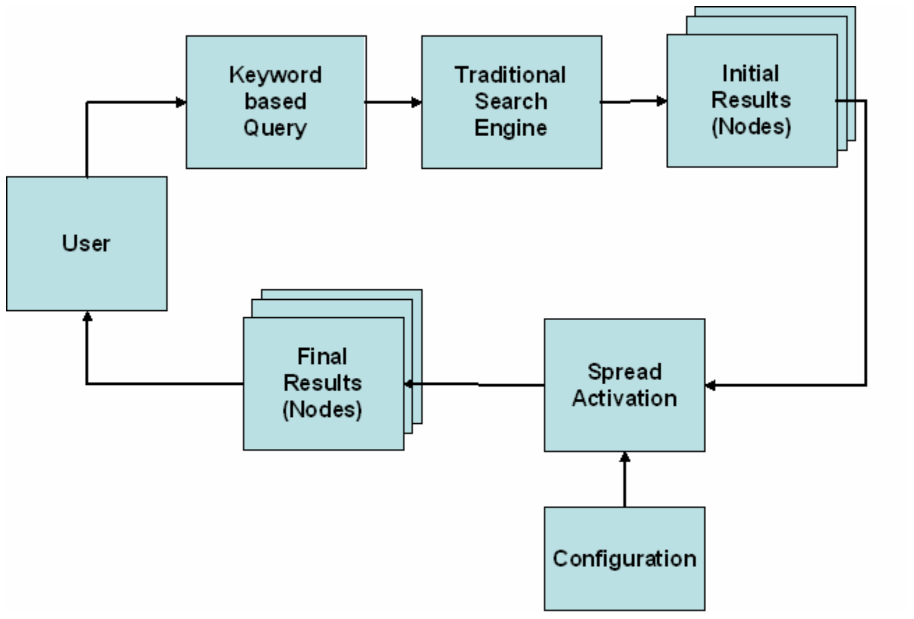


Figure 64: Architecture of the hybrid approach for Searching in the Semantic Web.

The picture starts on the left side of the user. As you can see, the user first enters his keywords as a query. Using a traditional search engine, all entries related to these terms are listed in the knowledge graph. Before the results are forwarded to the user, Spread Activation is used to search the Knowledge again for results that are related to the first results list and exceed a certain coverage level for the search terms.

Spread activation works by linking individual entries in the descriptions of testbed services and adding weights to these links. An algorithm for this technology then works through the knowledge graph with the results already found and selects all that activate the search function.

Finally, the original and newly found hits are delivered to the user. This approach is thus implemented in the prototypes.

Reference:

1. Ghijsen, M.; van der Ham, J.; Grosso, P.; de Laat, C. Towards an Infrastructure Description Language for Modeling Computing Infrastructures. 10th International Symposium on Parallel and Distributed Processing with Applications. IEEE, 2012, pp. 207–214.

2. Ghijsen, M.; van der Ham, J.; Grosso, P.; Dumitru, C.; Zhu, H.; Zhao, Z.; de Laat, C. A Semantic-Web Approach for Modeling Computing Infrastructures. Computers and Electrical Engineering 2013, 39, 2553–2565.

3. van der Ham, J.; Stéger, J.; Laki, S.; Kryftis, Y.; Maglaris, V.; de Laat, C. The NOVI information models. Future Generation Computer Systems 2015, 42, 64–73.

Co-funded by the Horizon 2020 Framework Programme of the European Union

4. Morsey, M.; Willner, A.; Loughnane, R.; Giatili, M.; Papagianni, C.; Baldin, I.; Grosso, P.; Al-Hazmi, Y. DBcloud: Semantic Dataset for the cloud. 2016 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS); IEEE: San Francisco, CA, 2016; pp. 207–212.

5. Cyganiak, R.; Wood, D.; Lanthaler, M. Resource Description Framework (RDF) 1.1 Concepts and Abstract Syntax. Recommendation, World Wide Web Consortium (W3C), 2014.

6. Beckett, D.; Berners-Lee, T.; Prud'hommeaux., E. Turtle-terse RDF triple language. Team submission, World Wide Web Consortium (W3C), 2008.

7. Lehmann, J.; Isele, R.; Jakob, M.; Jentzsch, A.; Kontokostas, D.; Mendes, P.N.; Hellmann, S.; Morsey, M.; van Kleef, P.; Auer, S.; Others. DBpedia-a large-scale, multilingual knowledge base extracted from wikipedia. Semantic Web Journal (SWJ) 2014, 5, 1–29.

8. Willner, Alexander, et al. "Using Semantic Web Technologies to Query and Manage Information within Federated Cyber-Infrastructures." *Data* 2.3 (2017): 21.

11 COLLABORATION WITH THE RAWFIE PROJECT ON ONTOLOGIES FOR UNMANNED VEHICLES AND SENSORS

RAWFIE (Road-, Air-, and Water- based Future Internet Experimentation, http://www.rawfie.eu) is a project funded by the European Commission (Horizon H2020 programme) under the Future Internet Research Experimentation (FIRE+) initiative that aims at providing research facilities for Internet of Things (IoT) devices. The project introduces a unique platform across the space and technology by integrating numerous test beds of unmanned vehicles for research experimentation in vehicular, aerial and maritime environments. The platform supports experimenters with smart tools to conduct and monitor experiments in the domains of IoT, networking, sensing and satellite navigation.

The SAMANT sub-project, with close collaboration with Fed4FIRE+, provides the appropriate tools and software enhancements at the RAWFIE testbed or federation level, to support functionalities related to resource discovery, booking and reservation, provisioning and release by experimenters, while addressing at the same time the corresponding authentication and authorization issues at the RAWFIE federation. Within the context of SAMANT, semantic information models are adopted for the description of UxVs, supporting the abovementioned functionalities in the federated RAWFIE environment.

11.1 SEMANTIC BASED RESOURCE DESCRIPTION

To our best knowledge, no previous work exists regarding the semantic description of the features of UxVs and their attached sensors. In a federated environment, such as the RAWFIE project, an in-depth description of UxVs and their equipped sensors would support user in all phases of an experiment, like resource discovery, reservation and construction of an execution scenario. Given the heterogeneity of the various UxVs employed in the case of RAWFIE, one particular issue that emerges is the description of these offerings. SFA, the de facto standard API for testbed federation, uses XML-based Resource Specifications (RSpecs) with arbitrary extensions to describe, discover, provision and release resources. However, such tree-based data models, lack consistency, standardized vocabularies as well as semantic meanings, therefore impede interoperability within a federation [WiPa15][MoWi16]. In the context of the SAMANT project, re-usage and extension of already well-defined standard semantic models are adopted for representing and linking RAWFIE federated resources. Additionally, the usage of a semantic registry repository for testbeds and resources will enable experimenters to find and book resources more easily. For this purpose, the OMN ontology suite [WiPa15], [MoWi16] is adopted and extended towards the semantic description of RAWFIE federated vehicular, aerial and maritime environment. The use of OMN leads to:

- Introduction of the necessary extensions and adoption of existing ontologies relative to the RAWFIE experimentation environment (UxVs, sensors, etc.),
- Maintenance of compatibility and interoperability with existing SFA-enabled infrastructures by using the information model and the corresponding data models with Aggregate Manager and MySlice components.

SAMANT ontology follows the common practice, in semantic modeling, of using already well-defined ontologies. SAMANT ontology is generic and aims to describe any UxV testbed apart

Co-funded by the Horizon 2020 Framework Programme of the European Union

from RAWFIE project requirements. It extends the OMN ontology suite and aims at describing RAWFIE UxVs resources and their embedded sensors. It adopts many concepts from the ontologies of OMN suite and adds to this ontology suite two new domain-specific ontology about unmanned vehicles and sensors, as it is shown in Figure 65. Furthermore, these new ontologies re-use already defined models from relevant ontologies on sensors and measurements.

11.1.1 SAMANT UxV Ontology

The extension of OMN ontology suite on the description of unmanned vehicles focuses on the semantic modeling of the UxVs resources, their reservation lifecycle (discovery, reservation and release) and the attributes of UxV testbeds and users. It consists of 59 classes, which are defined in the ontology or imported by others, 41 object properties that represent the relationships between the classes and 55 data properties that describe the features of the federated testbeds, the unmanned vehicles and the experimenters. Uxv class represents any kind of UxV and is descendant of the omn:Resource class of the upper omn-resource ontology.

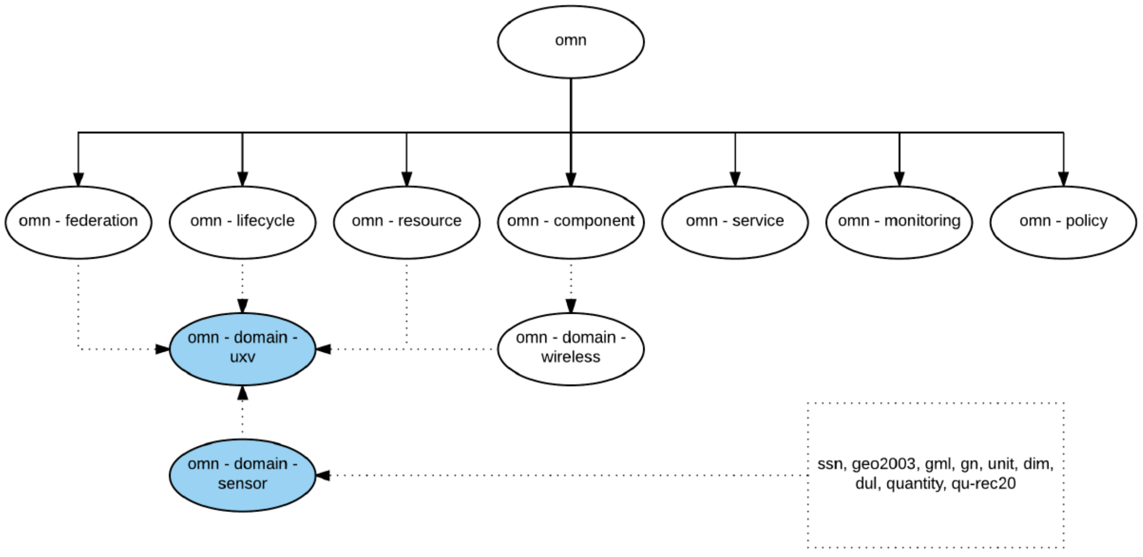


Figure 65: OMN suite

Data properties as Battery, Max Take off Weight, Speed, and Endurance etc. provide all the essential characteristic of UxVs. Every Uxv testbeds is defined by Testbed class and is descendant of Infrastructure class from omn-federation ontology. It connects with UxV and foaf:Person class with object properties, named :hasResource and :isTestbedOf respectively. All information about testbeds are represented by data properties such as name, description, countryCode, geo:alt, geo:long, geo:lat, which specifically provide geographical position of each testbed's area using the GeoRSS Feature Model and ontology [Geo07]. foaf:Person class defines every concept regarding the experimenters and administrator of the testbeds and is imported by the FoaF ontology.

Co-funded by the Horizon 2020 Framework Programme of the European Union

Several classes define essential concepts of UxVs attributes. Interface, Network Controller, Channel classes describe the characteristics of the communication interfaces of UxVs. HealthStastus class represents the current state of each UxV. ConfiParameters and ExperimentResourceConfig classes define the concepts of the configuration parameters of the unmanned vehicles and the experimental scenarios. As it is mentioned earlier, SAMANT ontology supports the reservation lifecycle of UxVs. For this purpose, many concepts from omn-lifecycle ontology are imported, which describe the entire lifecycle of resource management in federated testbeds. The reservation of a UxV is modeled by omn:Reservation and omn-lifecycle:Lease classes while the state of a reservation is defined by omn-lifecycle:ReservationState and its subclasses, named Allocated, Cancelled, Pending, Provisioned, Unallocated. The object properties omn:hasReservation and omn:isReservationOf expess the relationship between them. Table 11 summarizes all the data properties of SAMANT UxV ontology. Figure 66 shows the structure of SAMANT UxV ontology. The ontology - in Turtle format - is available at the following link:

https://github.com/w3c/omn/blob/master/omnlib/ontologies/unchecked/omn-domain-uxv.ttl

Table 11: UxV Attributes

Data Property Name	Type
Country Code	string
weight	double
Take Off Weight	double
length	double
width	double
height	double
diameter	double
endurance	integer
battery	integer
Channel Num*	integer
Lower Bound Frequency*	integer
Upper Bound Frequency*	integer
Antenna Count	integer
UxV Description	string
Testbed Description	string
ConfigParameters Description	string
Uav Support	boolean
Ugv Support	Boolean
Usv Support	boolean
Interface Vendor (communication)	string
Interface Nominal Bitrate	double
Latitude	double
Altitude	double
Longitude	double

Co-funded by the Horizon 2020 Framework Programme of the European Union

ConfigParametersMinValue	double
ConfigParametersMaxValue	Double

11.1.2 SAMANT Sensor Ontology

SAMANT Sensor ontology is an omn-domain specific ontology that models the embedded sensors on unmanned vehicles and aims at helping candidate users to find the suitable UxV with the appropriate sensors for his experiment. This ontology imports concepts from already well-defined ontologies about sensors and measurements, such as the SSN ontology of the W3C Semantic Sensor Networks Incubator Group (SSN-XG) [CoBa12] and ontology for quantity kinds and units [Lefo05]. SAMANT Sensor ontology - in Turtle format - is available at the following link:

https://github.com/w3c/omn/blob/master/omnlib/ontologies/unchecked/omn-domain-sensor.ttl

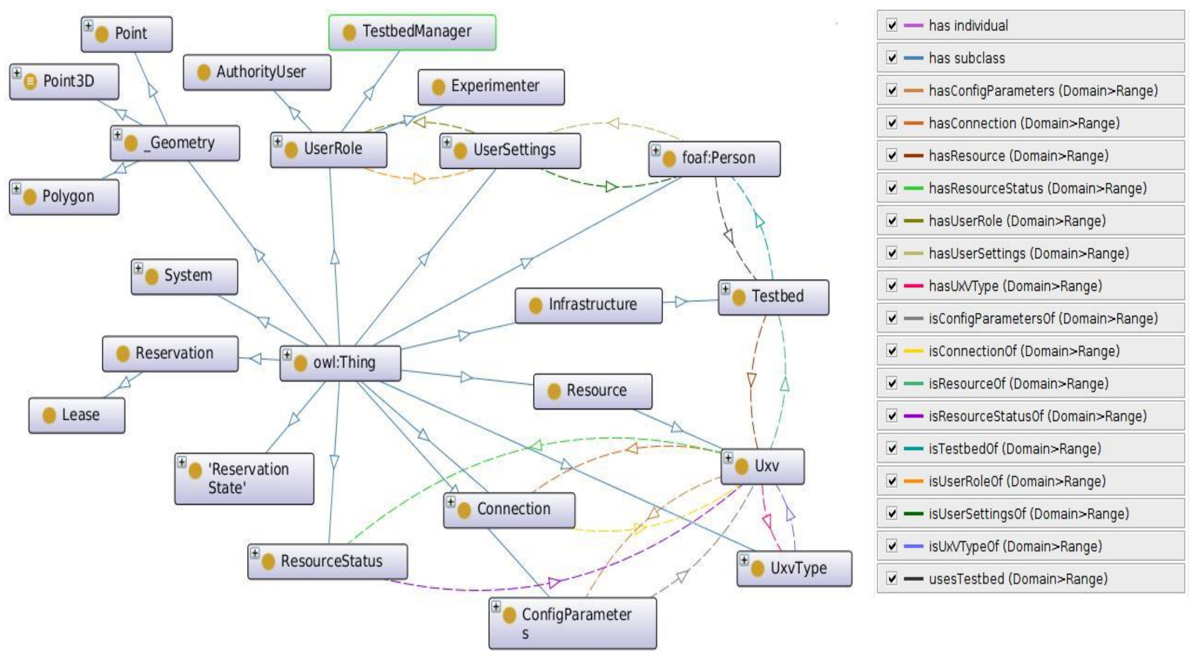

Point
TestbedManager
Point3D
AuthorityUser
Experimenter
_Geometry
UserRole
UserSettings
foaf:Person
Polygon
System
Infrastructure
Testbed
Reservation
owl:Thing
Resource
Lease
'Reservation State'
Uxv
Connection
ResourceStatus
UxvType
ConfigParameters
has individual
has subclass
hasConfigParameters (Domain>Range)
hasConnection (Domain>Range)
hasResource (Domain>Range)
hasResourceStatus (Domain>Range)
hasUserRole (Domain>Range)
hasUserSettings (Domain>Range)
hasUxVType (Domain>Range)
isConfigParametersOf (Domain>Range)
isConnectionOf (Domain>Range)
isResourceOf (Domain>Range)
isResourceStatusOf (Domain>Range)
isTestbedOf (Domain>Range)
isUserRoleOf (Domain>Range)
isUserSettingsOf (Domain>Range)
isUxVTypeOf (Domain>Range)
usesTestbed (Domain>Range)

Figure 66: OMN UxV ontology

Each UxV is equipped with several sensors and some of them are able to measure different phenomena simultaneously. For this reason, each UxV has a root multi-sensor system that contains all underlying individual and multiple sensors. The concepts of sensors and multi-sensor systems are defined by the ssn:SensingDevice and ssn:System classes respectively. The relation between a UxV and its multi-sensor system is expressed by hasSensorSystem and isSensorSystemOf object properties. ssn:hasSubSystem object property links every multi-sensor system with the underlying individual sensors. Additionally, many data properties provide information about single sensors and multi-sensor systems as it shown in Table 12. Every sensor is associated with a measuring property and one or more measuring units, e.g. quantity:tempareture, unit:kelvin and unit:degreeCelcius. These concepts are denoted by

Co-funded by the Horizon 2020 Framework Programme of the European Union

qu:QuantityKind and qu:Unit classes which model a large number of physical quantity (i.e. mass, pressure, velocity, electrical current etc) and the corresponding units of measurement and they are imported by W3C ontology for quantity kinds and units [Lefo05].

Finally the 'Feature of Interest' concept is an abstraction of real world phenomena, which includes air, ground and water concepts and it is defined by ssn:FeatureOfInterest class. Every physical quantity can be property of one or more of the three 'Feature of Interest' concepts. This relation is expressed by ssn:isPropertyOf and ssn:hasProperty object properties. Figure 67 shows the structure of SAMANT Sensor ontology.

Table 12: Sensor Attributes

Sensor Attribute	Type
Product Name	string
Vendor Name	string
Serial	string
Description	string
Observing Property	string
Unit of Measurement	string

Co-funded by the Horizon 2020 Framework Programme of the European Union

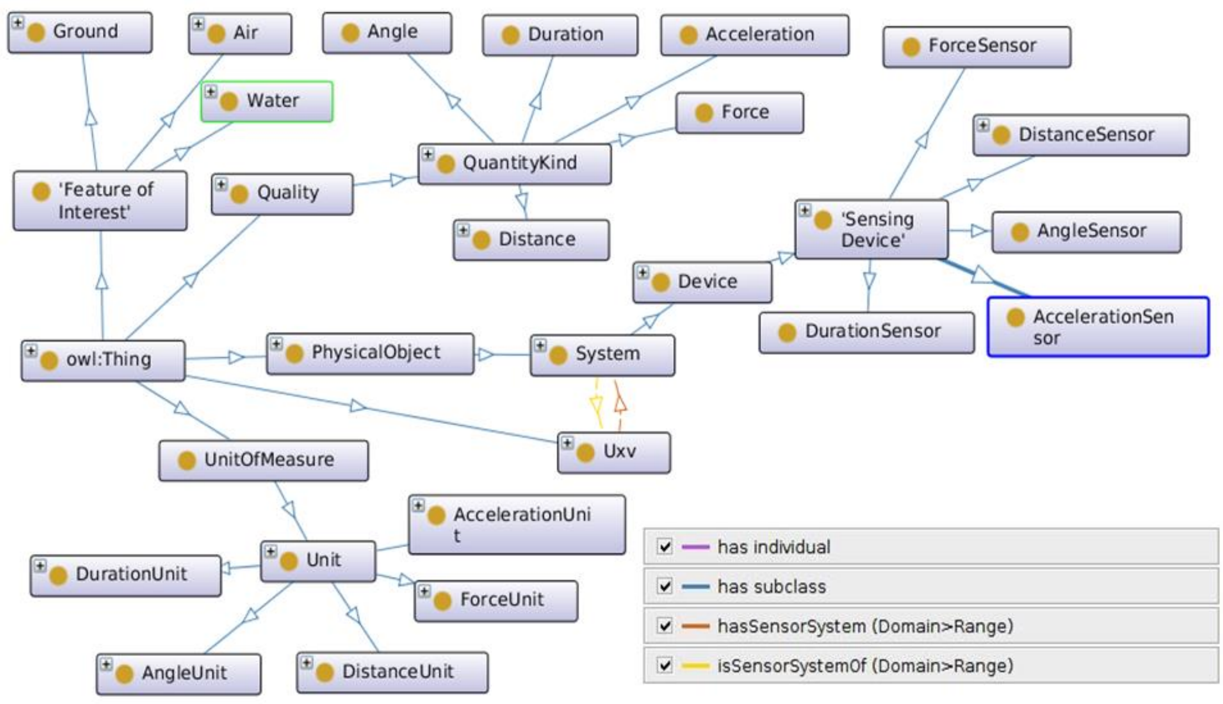


Figure 67: OMN Sensor Ontology

11.2 REFERENCES

[Lefo05]	Lefort L. Ontology for quantity kinds and units: units and quantities definitions. W3 Semantic Sensor Network Incubator Activity. 2005.
[MoWi16]	M. Morsey, A. Willner, R. Loughnane, M. Giatili, C. Papagianni, I. Baldin, P. Grosso, Y. Al-Hazmi, “DBcloud: Semantic Dataset for the Cloud”, accepted to appear at CRNET, Infocom 2016.
[CoBa12]	Compton, M., Barnaghi, P., Bermudez, L., GarcíA-Castro, R., Corcho, O., Cox, S.,... & Huang, V. (2012). The SSN ontology of the W3C semantic sensor network incubator group. Web Semantics: Science, Services and Agents on the World Wide Web, 17, 25-32.
[WiPa15]	A. Willner, C. Papagianni, M. Giatili, P. Grosso, M. Morsey, Al-Hazmi Y., I. Baldin, "The Open-Multinet Upper Ontology - Towards the Semantic-based Management of Federated Infrastructures", The 10th International Conference on Testbeds and

Co-funded by the Horizon 2020 Framework Programme of the European Union

	Research Infrastructures for the Development of Networks & Communities (TRIDENTCOM 2015), Vancouver, Canada, June 2015.

Co-funded by the Horizon 2020 Framework Programme of the European Union

12 APPENDIX : SLA COMPONENT FIRST CYCLE DOCUMENTATION

12.1 API DOCUMENTATION

12.1.1 API Introduction

The REST interface to the sla-core system has the following conventions:

- Every entity is created with a POST to the collection url. The body request contains the serialized entity in the format specified in the content-type header. The location header of the response refers to the url of the new allocated resource. The return code is a 201 on success. Templates and agreements have special considerations (see the corresponding section).
- A query for an individual item is a GET to the url of the resource (collection url + external id). The format of the response is specified in the http header with the accept parameter. The return code is 200. As expected, a not found resource returns a 404.
- Any other query is usually a GET to the collection's url, using the GET parameters as the query parameters. The result is a list of entities that match the parameters, despite the actual number of entities. The return code is 200, even if the list is empty.
- Any unexpected error processing the request returns a 5xx.
- An entity (or list) is serialized in the response body by default with the format specified in the Content-type header (if specified). The request may have an Accept header, that will be used if the resource allows more than one Content-type.
- Updating an entity involves a PUT request, with the corresponding resource serialized in the body in the format specified in the content-type header. The return code is 200.
- If a query has begun and/or end parameters, the following search is done: begin <= entity date < end

12.1.2 Generic operations

The generic operations of resources are shown below. Each particular resource (in following sections) shows the supported operations and any deviation from the behavior of generic operations.

The format of a resource can be modified by a project by using serializers.

- **GET /{resources}/{uuid}** Retrieve an entity by its uuid.

Request

```
GET /resources/{uuid} HTTP/1.1
```

Response in XML

```
HTTP/1.1 200 OK

Content-Type: application/xml



<?xml version="1.0" encoding="UTF-8"?>

<resource>...</resource>
```

Response in JSON

```
HTTP/1.1 200 OK

Content-Type: application/json



{ ... }
```

Usage (for JSON and XML)

```
curl   -H  "Accept:  application/xml"  http://localhost:8080/sla-service/resources/fc923960-
03fe-41

curl   -H  "Accept:  application/json"  http://localhost:8080/sla-service/resources/fc923960-
03fe-41
```

- **GET /resources{?param1=value1&param2=value2...}**

Search the resources that fulfill the params. All resources are returned if there are no parameters.

Request

```
GET /resources?param1=value1 HTTP/1.1
```

Response in XML

```
HTTP/1.1 200 OK

Content-type: application/xml



<?xml version="1.0" encoding="UTF-8"?>

<resources>

  <resource>...</resource>

  <resource>...</resource>

  <resource>...</resource>

<resources/>
```

Response in JSON

```
HTTP/1.1 200 OK

Content-type: application/json



[{...},{...},{...}]
```

Usage (for JSON and XML)

```
curl [-X GET] -H "Accept: application/xml" localhost:8080/sla-service/resources

curl [-X GET] -H "Accept: application/xml" localhost:8080/sla-service/resources?name=res-name

curl [-X GET] -H "Accept: application/json" localhost:8080/sla-service/resources

curl [-X GET] -H "Accept: application/json" localhost:8080/sla-service/resources?name=res-name
```

12.1.2.1 POST /resources

Create a new resource. The created resource will be accessed by its uuid. A message will be the usual response.

Request in XML

Co-funded by the Horizon 2020
Framework Programme of the European Union

```
POST /resources HTTP/1.1

Content-type: application/xml



<resource>...</resource>
```

Request in JSON

```
POST /resources HTTP/1.1

Content-type: application/json



{...}
```

Usage (for JSON and XML)

```
curl -H "Accept: application/xml" -H "Content-type: application/xml" -d@<filename> -X POST
localhost:8080/sla-service/resources

curl -H "Accept: application/json" -H "Content-type: application/json" -d@<filename> -X POST
localhost:8080/sla-service/resources
```

12.1.2.2 UPDATE /resources/{uuid}

Updates an existing resource. The content in the body will overwrite the content of the resource. The uuid in the body must match the one from the url o not being informed.

Request in XML

```
PUT /resources/{uuid} HTTP/1.1

Content-type: application/xml



<resource>...</resource>
```

Request in JSON

```
PUT /resources/{uuid} HTTP/1.1
```

Co-funded by the Horizon 2020
Framework Programme of the European Union

```
Content-type: application/xml



{...}
```

Response in XML

```
HTTP/1.1 200 Ok

Content-type: application/xml



<?xml version="1.0" encoding="UTF-8"?>

<resource>

    ...

</resource>
```

Response in JSON

```
HTTP/1.1 200 Ok

Content-type: application/json



{...}
```

Usage

```
curl  -H "Accept: application/xml" -H "Content-type: application/xml" -d@<filename> -X PUT
localhost:8080/sla-service/resources/{uuid}

curl  -H "Accept: application/json" -H "Content-type: application/json" -d@<filename> -X PUT
localhost:8080/sla-service/resources/{uuid}
```

12.1.2.3 DELETE /resources/{uuid}

Deletes an existing resource.

Request

Co-funded by the Horizon 2020
Framework Programme of the European Union

```
DELETE /providers/{uuid} HTTP/1.1
```

Response in XML and JSON

```
HTTP/1.1 200 Ok

Content-type: application/[xml | json]



... (free text indicating that the resource has been removed)
```

Usage (for JSON and XML)

```
curl -H "Accept: application/xml" -X DELETE localhost:8080/sla-service/resources/fc923960-
03fe-41

curl -H "Accept: application/json" -X DELETE localhost:8080/sla-service/resources/fc923960-
03fe-41
```

12.1.2.4 Messages

Some of the above mentioned methods might return a message. Messages can be returned as XML or JSON.

Message Response in XML

```
Content-type: application/xml



<?xml version="1.0" encoding="UTF-8"?>

<message code="xxx" elemendId="..." message="..."/>
```

Message Request in JSON

```
Content-type: application/json



{"code":"xxx", "elemendId":..., "message": ...}
```

Co-funded by the Horizon 2020
Framework Programme of the European Union

Co-funded by the Horizon 2020
Framework Programme of the European Union

12.1.3 Providers

- Provider collection URI: /providers
- Provider URI: /providers/{uuid}

A provider is serialized in XML as:

```
<provider>
   <uuid>fc923960-03fe-41eb-8a21-a56709f9370f</uuid>
   <name>provider-prueba</name>
</provider>
```

A provider is serialized in JSON as:

```
{"uuid":"fc923960-03fe-41eb-8a21-a56709f9370f",
 "name":"provider-prueba"}
```

- **GET /providers/{uuid}** Retrieves a specific provider identified by uuid

Error message:

- 404 is returned when the uuid doesn't exist in the database.

- **GET /providers** Retrieves the list of all providers

- **POST /providers** Creates a provider. The uuid is in the file beeing send

Error message:

- 409 is returned when the uuid or name already exists in the database.

- **DELETE /providers/{uuid}** Removes the provider identified by uuid.

Error message:

- 404 is returned when the uuid doesn't exist in the database.
- 409 is returned when the provider code is used.

Co-funded by the Horizon 2020 Framework Programme of the European Union

12.1.4 Templates

- Templates collection URI: /templates
- Template URI: /templates/{TemplateId}

The TemplateId matches the TemplateId attribute of wsag:Template element when the template is created. A template is serialized in XML as defined by ws-agreement.
An example of template in XML is:

```
<?xml version="1.0" encoding="UTF-8"?>
    <wsag:Template                   xmlns:wsag="http://www.ggf.org/namespaces/ws-agreement"
xmlns:sla="http://sla.atos.eu"
    wsag:TemplateId="template012">
    <wsag:Name>ExampleTemplate</wsag:Name>
        <wsag:Context>
            <wsag:AgreementInitiator>provider02</wsag:AgreementInitiator>
            <wsag:ServiceProvider>provider01</wsag:ServiceProvider>
            <wsag:ExpirationTime>2014-03-07T12:00:00+0100</wsag:ExpirationTime>
            <wsag:ServiceProvider>AgreementInitiator</wsag:ServiceProvider>
            <wsag:TemplateId>template01</wsag:TemplateId>
            <sla:Service xmlns:sla="http://sla.atos.eu">service3</sla:Service>
        </wsag:Context>
        <wsag:Terms>
            <wsag:All>
            <!-- functional description -->
                <wsag:ServiceDescriptionTerm                             wsag:Name="General"
wsag:ServiceName="Service0001">A GPS service</wsag:ServiceDescriptionTerm>
                <wsag:ServiceDescriptionTerm                   wsag:Name="GetCoordsOperation"
wsag:ServiceName="GPSService0001">operation      to       call       to       get       the
coords</wsag:ServiceDescriptionTerm>
                <!-- domain specific reference to a service (additional or optional to SDT)
-->
                <wsag:ServiceReference                               wsag:Name="CoordsRequest"
wsag:ServiceName="GPSService0001">
                    <wsag:EndpointReference>

<wsag:Address>http://www.gps.com/coordsservice/getcoords</wsag:Address>
                        <wsag:ServiceName>gps:CoordsRequest</wsag:ServiceName>
                    </wsag:EndpointReference>
                </wsag:ServiceReference>
                <wsag:ServiceProperties                  wsag:Name="AvailabilityProperties"
wsag:ServiceName="GPS0001">
                    <wsag:Variables>
                        <wsag:Variable                               wsag:Name="ResponseTime"
wsag:Metric="metric:Duration">
                            <wsag:Location>qos:ResponseTime</wsag:Location>
                        </wsag:Variable>
                    </wsag:Variables>
                </wsag:ServiceProperties>
            <wsag:ServiceProperties                          wsag:Name="UsabilityProperties"
wsag:ServiceName="GPS0001">
                <wsag:Variables>
                    <wsag:Variable                                 wsag:Name="CoordDerivation"
wsag:Metric="metric:CoordDerivationMetric">
                        <wsag:Location>qos:CoordDerivation</wsag:Location>
                    </wsag:Variable>
                </wsag:Variables>
```

Co-funded by the Horizon 2020
Framework Programme of the European Union

```
            </wsag:ServiceProperties>
            <!-- statements to offered service level(s) -->
            <wsag:GuaranteeTerm wsag:Name="FastReaction" wsag:Obligated="ServiceProvider">
                <wsag:ServiceScope wsag:ServiceName="GPS0001">
                    http://www.gps.com/coordsservice/getcoords
                </wsag:ServiceScope>
                <wsag:QualifyingCondition>
                    applied when current time in week working hours
                </wsag:QualifyingCondition>
                <wsag:ServiceLevelObjective>
                    <wsag:KPITarget>
                        <wsag:KPIName>FastResponseTime</wsag:KPIName>
                        <wsag:Target>
                            //Variable/@Name="ResponseTime" LOWERTHAN 1 second
                        </wsag:Target>
                    </wsag:KPITarget>
                </wsag:ServiceLevelObjective>
            </wsag:GuaranteeTerm>
        </wsag:All>
    </wsag:Terms>
</wsag:Template>
```

An example of template in JSON is:

```
{
    "templateId":"template05",
    "context":{
        "agreementInitiator":"provider02",
        "agreementResponder":null,
        "serviceProvider":"AgreementInitiator",
        "templateId":"template01",
        "service":"service3",
        "expirationTime":"2014-03-07T12:00:00CET"
    },
    "name":"ExampleTemplate",
    "terms":{
        "allTerms":{
            "serviceDescriptionTerm":{
                "name":null,
                "serviceName":null
            },
            "serviceProperties":[
                {"name":null, "serviceName":null, "variableSet":null},
                {"name":null, "serviceName":null, "variableSet":null}
            ],
            "guaranteeTerms":[
                {
                "name":"FastReaction",
                "serviceScope":{
                        "serviceName":"GPS0001",
                        "value":"http://www.gps.com/coordsservice/getcoords"
                },
                "serviceLevelObjetive":{
                        "kpitarget":{
                            "kpiName":"FastResponseTime",
                            "customServiceLevel":null
                        }
                    }
```

Co-funded by the Horizon 2020
Framework Programme of the European Union

```
                }
            ]
        }
    }
}
```

- **GET /templates/{TemplateId}** Retrieves a template identified by TemplateId.

Error message:
  - 404 is returned when the uuid doesn't exist in the database.

- **GET /templates{?serviceIds,providerId}**

The parameters are:
  - serviceIds: string with coma separated values (CSV) with the id's of service that is associated to the template
  - providerId: id of the provider that is offering the template

- **POST /templates** Creates a new template. The file might include a TemplateId or not. In case of not beeing included, a uuid will be assigned.

Error message:
  - 409 is returned when the uuid already exists in the database.
  - 409 is returned when the provider uuid specified in the template doesn't exist in the database.
  - 500 when incorrect data has been suplied

- **PUT /templates/{TemplateId}** Updates the template identified by TemplateId. The body might include a TemplateId or not. In case of including a TemplateId in the file, it must match with the one from the url.

Error message:
  - 409 when the uuid from the url doesn't match with the one from the file or when the system has already an agreement associated
  - 409 when template has agreements associated.
  - 409 provider doesn't exist
  - 500 when incorrect data has been suplied

- **DELETE /templates/{TemplateId}** Removes the template identified by TemplateId.

Error message:
  - 409 when agreements are still associated to the template
  - 404 is returned when the uuid doesn't exist in the database.

Co-funded by the Horizon 2020 Framework Programme of the European Union

12.1.5 Agreements

- Agreements collection URI: /agreements
- Agreement URI: /agreement/{AgreementId}

The AgreementId matches the AgreementId attribute of wsag:Agreement element when the agreement is created. An example of agreement in XML is:

```
<?xml version="1.0" encoding="UTF-8"?>
<wsag:Agreement                     xmlns:wsag="http://www.ggf.org/namespaces/ws-agreement"
xmlns:sla="http://sla.atos.eu">
    <wsag:Name>ExampleAgreement</wsag:Name>
    <wsag:Context>
        <wsag:ExpirationTime>2014-03-07T12:00:00+0100</wsag:ExpirationTime>
        <wsag:AgreementInitiator>RandomClient</wsag:AgreementInitiator>
        <wsag:AgreementResponder>provider03</wsag:AgreementResponder>
        <wsag:ServiceProvider>AgreementResponder</wsag:ServiceProvider>
        <wsag:TemplateId>template04</wsag:TemplateId>
        <sla:Service>service01</sla:Service>
    </wsag:Context>
    <wsag:Terms>
        <wsag:All>
            <wsag:ServiceProperties                              wsag:Name="NonFunctional"
wsag:ServiceName="ServiceName">
                <wsag:Variables>
                    <wsag:Variable wsag:Name="ResponseTime" wsag:Metric="xs:double">
                        <wsag:Location>qos:ResponseTime</wsag:Location>
                    </wsag:Variable>
                </wsag:Variables>
            </wsag:ServiceProperties>
            <wsag:GuaranteeTerm wsag:Name="GTResponseTime">
                <wsag:ServiceScope wsag:ServiceName="ServiceName" />
                <wsag:ServiceLevelObjective>
                    <wsag:KPITarget>
                        <wsag:KPIName>ResponseTime</wsag:KPIName>
                        <wsag:CustomServiceLevel>{"constraint"     :     "ResponseTime     LT
100"}</wsag:CustomServiceLevel>
                    </wsag:KPITarget>
                </wsag:ServiceLevelObjective>
            </wsag:GuaranteeTerm>
        </wsag:All>
    </wsag:Terms>
</wsag:Agreement>
```

An example of agreement in JSON is:

```
{
    "agreementId":"agreement07",
    "name":"ExampleAgreement",
    "context":{
        "agreementInitiator":"client-prueba",
        "expirationTime":"2014-03-07T12:00:00+0100",
        "templateId":"template02",
        "service":"service5",
        "serviceProvider":"AgreementResponder",
        "agreementResponder":"provider03"
```

Co-funded by the Horizon 2020
Framework Programme of the European Union

```
    },
    "terms": {
        "allTerms":{
            "serviceDescriptionTerm":null,
            "serviceProperties":[
                {
                    "name":"ServiceProperties",
                    "serviceName":"ServiceName",
                    "variableSet":{
                    "variables":[
                        { "name":"metric1","metric":"xs:double","location":"metric1"},
                        { "name":"metric2","metric":"xs:double","location":"metric2"},
                        { "name":"metric3","metric":"xs:double","location":"metric3"},
                        { "name":"metric4","metric":"xs:double","location":"metric4"}
                    ]
                }
            }
        ],
        "guaranteeTerms":[
            {
                "name":"GTMetric1",
                "serviceScope":{"serviceName":"ServiceName","value":""},
                "serviceLevelObjetive":{
                    "kpitarget":{
                        "kpiName":"metric1",
                        "customServiceLevel":"{\"constraint\"  :  \"metric1  BETWEEN  (0.05,
1)\"}"
                    }
                }
            },{
                "name":"GTMetric2",
                "serviceScope":{"serviceName":"ServiceName","value":""},
                "serviceLevelObjetive":{
                    "kpitarget":{
                        "kpiName":"metric2",
                        "customServiceLevel":"{\"constraint\"  :  \"metric2  BETWEEN  (0.1,
1)\"}"
                    }
                }
            },{
                "name":"GTMetric3",
                "serviceScope":{"serviceName":"ServiceName","value":""},
                "serviceLevelObjetive":{
                    "kpitarget":{
                        "kpiName":"metric3",
                        "customServiceLevel":"{\"constraint\"  :  \"metric3  BETWEEN  (0.15,
1)\"}"
                    }
                }
            },{
                "name":"GTMetric4",
                "serviceScope":{"serviceName":"ServiceName","value":""},
                "serviceLevelObjetive":{
                    "kpitarget":{
                        "kpiName":"metric4",
                        "customServiceLevel":"{\"constraint\"  :  \"metric4  BETWEEN  (0.2,
1)\"}"
                    }
                }
```

Co-funded by the Horizon 2020
Framework Programme of the European Union

```
            }
        ]
    }
}
```

- **GET /agreements/{AgreementId}** Retrieves an agreement identified by AgreementId.

Error message:
  - 404 is returned when the uuid doesn't exist in the database.

- **GET /agreements/** Retrieves the list of all agreements.

- **GET /agreements{?consumerId,providerId,templateId,active}**

The parameters are:
  - consumerId: uuid of the consumer (value of Context/AgreementInitiator if Context/ServiceProvider equals "AgreementResponder").
  - providerId: uuid of the provider (value of Context/AgreementResponder if Context/ServiceProvider equals "AgreementResponder")
  - templateId: uuid of the template the agreement is based on.
  - active: boolean value (value in {1,true,0,false}); if true, agreements currently enforced are returned.

- **GET /agreementsPerTemplateAndConsumer{?consumerId,templateUUID}**

The parameters are:
  - consumerId: uuid of the consumer (value of Context/AgreementInitiator if Context/ServiceProvider equals "AgreementResponder").
  - templateUUID: uuid of the template in which the agreement is based

- **POST /agreements** Creates a new agreement. The body might include a AgreementId or not. In case of not being included, a uuid will be assigned. A disabled enforcement job is automatically created.

Error message:
  - 409 is returned when the uuid already exists in the database
  - 409 is returned when the provider uuid specified in the agreement doesn't exist in the database
  - 409 is returned when the template uuid specified in the agreement doesn't exist in the database
  - 500 when incorrect data has been suplied.

- **DELETE /agreements/{AgreementId}** Removes the agreement identified by AgreementId.

Error message:
  - 404 is returned when the uuid doesn't exist in the database

Co-funded by the Horizon 2020 Framework Programme of the European Union

- **GET /agreements/active** Returns the list of active agreements.

- **GET /agreements/{AgreementId}/context** Only the context from the agreement identified by AgreementId is returned.

Error message:

- 404 is returned when the uuid doesn't exist in the database
- 500 when the data agreement was recorded incorrectly and the data cannot be supplied

Request in XML

```
GET -H "Accept: application/xml" /agreements/{agreement-id}/context HTTP/1.1
```

Request in JSON

```
GET -H "Accept: application/json" /agreements/{agreement-id}/context HTTP/1.1
```

Response in XML

```
HTTP/1.1 200 OK

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<wsag:Context  xmlns:sla="http://sla.atos.eu"  xmlns:wsag="http://www.ggf.org/namespaces/ws-
agreement">
    <wsag:AgreementInitiator>RandomClient</wsag:AgreementInitiator>
    <wsag:AgreementResponder>provider02</wsag:AgreementResponder>
    <wsag:ServiceProvider>AgreementResponder</wsag:ServiceProvider>
    <wsag:ExpirationTime>2014-03-07T12:00:00CET</wsag:ExpirationTime>
    <wsag:TemplateId>template02</wsag:TemplateId>
    <sla:Service>service02</sla:Service>
</wsag:Context>
```

Response in JSON

```
HTTP/1.1 200 OK

{"AgreementInitiator":"RandomClient",
 "AgreementResponder":"provider02",
 "ServiceProvider":"AgreementResponder",
 "ExpirationTime":"2014-03-07T12:00:00CET",
 "TemplateId":"template02",
 "Service":"service02"}
```

Usage (for JSON and XML)

```
curl       -H       "Accept:       application/xml"       http://localhost:8080/sla-
service/agreements/agreement01/context
curl       -H       "Accept:       application/json"      http://localhost:8080/sla-
service/agreements/agreement01/context
```

Co-funded by the Horizon 2020 Framework Programme of the European Union

- **GET /agreements/{AgreementId}/guaranteestatus** Gets the information of the status of the different Guarantee Terms of an agreement.

There are three available states: NON_DETERMINED, FULFILLED, VIOLATED.
Error message:

- 404 is returned when the uuid doesn't exist in the database

Request in XML

```
GET -H "Accept: application/xml" /agreements/{agreement-id}/guaranteestatus
HTTP/1.1
```

Request in JSON

```
GET -H "Accept: application/json" /agreements/{agreement-id}/guaranteestatus
HTTP/1.1
```

Response in XML

```
HTTP/1.1 200 OK
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<guaranteestatus AgreementId="agreement02" value="FULFILLED">
    <guaranteetermstatus name="GTResponseTime" value="FULFILLED"/>
    <guaranteetermstatus name="GTPerformance" value="FULFILLED"/>
</guaranteestatus>
```

Response in JSON

```
HTTP/1.1 200 OK
{"AgreementId":"agreement02",
 "guaranteestatus":"FULFILLED",
 "guaranteeterms":
    [{"name":"GTResponseTime", "status":"FULFILLED"},
     {"name":"GTPerformance", "status":"FULFILLED"}]
 }
```

Usage (for JSON and XML)

```
curl        -H        "Accept:        application/xml"        http://localhost:8080/sla-
service/agreements/agreement01/guaranteestatus
curl        -H        "Accept:        application/json"       http://localhost:8080/sla-
service/agreements/agreement01/guaranteestatus
```

Co-funded by the Horizon 2020
Framework Programme of the European Union

12.1.6 Enforcement Jobs

An enforcement job is the entity which starts the enforcement of the agreement guarantee terms. An agreement can be enforced only if an enforcement job, linked with it, has been previously created and started. An enforcement job is automatically created when an agreement is created, so there is no need to create one to start an enforcement.

- Enforcement jobs collection URI: /enforcements
- Enforcement job URI: /enforcements/{AgreementId}

An enforcement job is serialized in XML as:

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<enforcement_job>
    <agreement_id>agreement02</agreement_id>
    <enabled>true</enabled>
    <last_executed>2014-08-13T10:01:01CEST</last_executed>
</enforcement_job>
```

An enforcement job is serialized in JSON as:

```json
{"enabled":true,
 "agreement_id":"agreement02",
 "last_executed":"2014-08-13T10:01:01CEST"}
```

- **GET /enforcements/{AgreementId}** Retrieves an enforcement job identified by AgreementId.

Error message:
- 404 is returned when the uuid doesn't exist in the database

- **GET /enforcements** Retrieves the list of all enforcement job.

- **POST /enforcements** Creates and enforcement job. Not required anymore. The enforcement job is automatically generated when an agreement is created.

Error message:
- 409 is returned when an enforcement with that uuid already exists in the database
- 404 is returnes when no agreement with uuid exists in the database

PUT /enforcements/{AgreementId}/start Starts an enforcement job.
Error message:
- 403 is returned when it was not possible to start the job

Request

```
PUT /enforcements/{agreement-id}/start HTTP/1.1
```

Response in XML and JSON

Co-funded by the Horizon 2020 Framework Programme of the European Union

```
HTTP/1.1 200 Ok
Content-type: application/[xml | json]

The enforcement job with agreement-uuid {agreement-id} has started
```

Usage (for JSON and XML)

```
curl -H "Accept: application/xml" -X PUT localhost:8080/sla-service/enforcements/fc923960-
03fe-41/start
curl -H "Accept: application/json" -X PUT localhost:8080/sla-service/enforcements/fc923960-
03fe-41/start
```

- **PUT /enforcements/{AgreementId}/stop** Stops an enforcement job

Error message:

- 403 is returned when it was not possible to start the job

Request

```
PUT /enforcements/{agreement-id}/stop HTTP/1.1
```

Response in XML and JSON

```
HTTP/1.1 200 OK
Content-type: application/[xml | json]

The enforcement job with agreement-uuid {agreement-id} has stoppped
```

Usage (for JSON and XML)

```
curl -H "Accept: application/xml" -X PUT localhost:8080/sla-service/enforcements/fc923960-
03fe-41/stop
curl -H "Accept: application/json" -X PUT localhost:8080/sla-service/enforcements/fc923960-
03fe-41/stop
```

Co-funded by the Horizon 2020
Framework Programme of the European Union

12.1.7 Violations

- Violations collection URI: /violations
- Violation URI: /violations/{uuid}

A violation is serialized in XML as:

```
<violation>
    <uuid>ce0e148f-dfac-4492-bb26-ad2e9a6965ec</uuid>
    <contract_uuid>agreement04</contract_uuid>
    <service_scope></service_scope>
    <metric_name>Performance</metric_name>
    <datetime>2014-08-13T10:01:01CEST</datetime>
    <actual_value>0.09555700123360344</actual_value>
</violation>
```

A violation is serialized in JSON as:

```
{"uuid":"e431d68b-86ac-4c72-a6db-939e949b6c1",
 "datetime":"2014-08-13T10:01:01CEST",
 "contract_uuid":"agreement07",
 "service_name":"ServiceName",
 "service_scope":"",
 "metric_name":"time",
 "actual_value":"0.021749629938806803"}
```

- **GET /violations/{uuid}** Retrieves information from a violation identified by the uuid.

- **GET /violations{?agreementId,guaranteeTerm,providerId,begin,end}**

Parameters:

- agreementId: if specified, search the violations of the agreement with this agreementId,
- guaranteeTerm: if specified, search the violations of the guarantee term with this name (GuaranteeTerm[@name]),
- providerId: if specified, search the violations raised by this provider.
- begin: if specified, set a lower limit of date of violations to search. Date format: yyyy-MM-dd'T'HHss
- end: if specified, set an upper limit of date of violations to search. Date format: yyyy-MM-dd'T'HHss

Error message:

- 404 when erroneous data is provided in the call

Co-funded by the Horizon 2020 Framework Programme of the European Union

12.1.8 Penalties

- Penalties collection URI: /penalties
- Penalty URI: /penalties/{uuid}

A penalty is serialized in XML as:

```
<penalty    xmlns:sla="http://sla.atos.eu"    xmlns:wsag="http://www.ggf.org/namespaces/ws-
agreement">
    <uuid>ec7fd8ec-d917-49a2-ad80-80ff9aa8269c</uuid>
    <agreement>agreement-a</agreement>
    <datetime>2015-01-21T18:42:00CET</datetime>
    <definition type="discount" expression="35" unit="%" validity="P1D"/>
</penalty>
```

A penalty is serialized in JSON as:

```
{
    "uuid":"bfc4bc66-d647-453a-b813-d130f6116daf",
    "datetime":"2015-01-21T18:49:00CET",
    "definition":{
        "type":"discount",
        "expression":"35",
        "unit":"%",
        "validity":"P1D"
    },
    "agreement":"agreement-a"
}
```

- **GET /penalties/{uuid}** Retrieves information from a penalty identified by the uuid.

- **GET /penalties{?agreementId,guaranteeTerm,begin,end}**

Parameters:

- agreementId: if specified, search the penalties of the agreement with this agreementId,
- guaranteeTerm: if specified, search the penalties of the guarantee term with this name (GuaranteeTerm[@name]),
- begin: if specified, set a lower limit of date of penalties to search,
- end: if specified, set an upper limit of date of penalties to search.

Error message:

- 404 when erroneous data is provided in the call

Co-funded by the Horizon 2020 Framework Programme of the European Union

12.2 INSTALLATION GUIDE

12.2.1 Requirements

The requirements to install a working copy of the sla core are:

- Oracle JDK >=1.6
- Database to install the database schema for the service: Mysql>=5.0
- Maven >= 3.0

12.2.2 Installation

Source of the Project

Download the project, for the moment the code is into the zip file, until we identify the source repository in the project.

- Unzip newVersionCoreSLA.zip

Delivered versions

- V0.1 installationGuide_SLA_v0.1.zip Basic version, without configuration parameters and basic API monitoring (without aggregated data)
- V0.2 installationGuide_SLA_v0.2.zip this version is integrated with the last monitoring data, which includes the aggregated data, moreover it include the configuration properties to be deployed in different nodes.
- V0.3 installationGuide_SLA_v0.2.zip, this version includes the common structure of the monitoring engine for NTUA and NITOS testbeds. It is included the raw data and the aggregated data (for multiple nodes).

Creating the mysql database

From mysql command tool, create a database (with a user with sufficient privileges, as root):

```
$ mysql -p -u root

mysql> CREATE DATABASE atossla;
```

Create a user:

```
mysql> CREATE USER atossla@localhost IDENTIFIED BY '_atossla_';
```

Co-funded by the Horizon 2020 Framework Programme of the European Union

```
mysql> GRANT ALL PRIVILEGES ON atossla.* TO atossla@localhost; -- * optional WITH
GRANT OPTION;
```

From command prompt, create needed tables:

```
$ mvn test exec:java -f sla-repository/pom.xml
```

Another option to create the database is execute a sql file from the project root directory:

```
$ bin/restoreDatabase.sh
```

The names used here are the default values of the sla core. See section configuration to know how to change the values.

12.2.3 Configuration

The project is made up of five main modules:

- SLA Repository
- SLA Enforcement
- SLA Service
- SLA Tools
- SLA Personalization

Several parameters can be configured through this *configuration.properties* file (which is placed in the parent directory).

1. db.* allows to configure the database username, password and name in case it has been changed from the proposed one in the section Creating the mysql database. It can be selected if queries from hibernate must be shown or not. These parameters can be overriden at deployment time through the use of environment variables (see section Running),
2. enforcement.* several parameters from the enforcement can be customized,
3. service.basicsecurity.* basic security is enabled. These parameters can be used to set the user name and password to access to the rest services.

Co-funded by the Horizon 2020 Framework Programme of the European Union

4. "parser.*" different parsers can be implemented for the agreement and template. By default, wsag standard parsers have been implemented and configured in the file. Also dateformat can be configured.
5. “monitoring.*” indicates the different parameters that should be configure in order to obtain the data from the monitoring system.
    1. url indicates the host name of the monitoring system
    2. token indicates the authentication token to connect with the monitoring system.

If you're creating the database using the command “*mvn test exec:java -f sla-repository/pom.xml”* please make sure that you configure properly sla-repository\src\main\resources\META-INF\persistence.xml. Make sure you're setting the username, password and connection url with the proper parameters.

```
<property name="hibernate.connection.username" value="atossla" />

<property name="hibernate.connection.password" value="_atossla_" />

<property                                    name="hibernate.connection.url"
value="jdbc:mysql://localhost:3306/atossla" />
```

12.2.4 Compiling

```
$ mvn install
```

If you want to skip tests:

```
$ mvn install -Dmaven.test.skip=true
```

The result of the command is a war in *sla-service/target*.

12.2.5 Running

runserver.sh script runs the sla-core server using jetty runner on port 8080 and / as context path.

Co-funded by the Horizon 2020 Framework Programme of the European Union

```
$ bin/runserver.sh
```

Some configuration parameters can be overridden using environment variables or jdk variables. The list of parameters overridable is:

- DB_DRIVER; default value is com.mysql.jdbc.Driver
- DB_URL; default value is jdbc:mysql://${db.host}:${db.port}/${db.name}
- DB_USERNAME; default value is ${db.username}
- DB_PASSWORD; default value is ${db.password}
- DB_SHOWSQL; default value is ${db.showSQL}
- MONITORING_URL; default value is ${monitoring.url}
- MONITORING_TOKEN; default value is ${monitoring.token}

For example, to use a different database configuration:

```
$ export DB_URL=jdbc:mysql://localhost:3306/sla

$ export DB_USERNAME=sla

$ export DB_PASSWORD=<secret>

$ export MONITORING_URL=http://vnews-2.netmode.ece.ntua.gr:3000



$ export MONITORING_TOKEN=Bearer <token>

$ bin/runserver.sh
```

12.2.6 Logging

By default, sla-core logs to stdout using log4j. The log4.properties file is stored in sla-service in sla-service/src/main/resources.

If you want to use another log4j configuration, you can pass a different properties file to the JRE using -Dlog4j.configuration=file:{file path}.

12.2.7 Testing

Check that everything is working:

Co-funded by the Horizon 2020
Framework Programme of the European Union

```
$ curl http://localhost:8080/api/providers
```

12.2.8 Adapters

We have created the adapters to get the data from the tested monitoring engine. For the moment, we have integrated Nitos and Ntua testbeds, which have the same structure messages. If other testbeds want to be integrated using the defined structure, it will not necessary to adapt anything. Nevertheless, if the new monitoring systems have another message definition, it will be necessary to create new adapters to integrate these new testbeds.

12.2.9 Security access

For the moment, we have activated two context to do the same.

- Without authentication

```
$ curl http://localhost:8080/api/providers
```

- With authentication

```
$ curl http://localhost:8080/api/secure/providers --user name:password
```

We can use for the moment without authentication since we need to change it for the authentication decided in the project. We can postpone this adaptation.

12.2.10 Running with other applications

If we want to deploy the component in other web application, it is only necessary to copy the generated war file to the folder of the Web Application. For example for the Tomcat tool:

cp <sla_core>/sla-service/target/sla-service.war <tomcat>/webapps

If we decide to maintain jetty, we will need to modify the bin/runserver.sh file to be executed in background and start/stop jetty tool as a service (to be decided).

Co-funded by the Horizon 2020 Framework Programme of the European Union