Control plane extension - Status of the SFA deployment

Thierry Parmentelat, Jordan Auge, Loïc Baron, Mohamed Amine Larabi, Nikos Mouratidis, Harris Niavis, Mohammed Yasin Rahman, Thierry Rakotoarivelo, Florian Schreiner, Donatos Stavropoulos, et al.

▶ **To cite this version:**

Thierry Parmentelat, Jordan Auge, Loïc Baron, Mohamed Amine Larabi, Nikos Mouratidis, et al.. Control plane extension - Status of the SFA deployment. 2013. hal-00948905

HAL Id: hal-00948905

https://hal.archives-ouvertes.fr/hal-00948905

Preprint submitted on 18 Feb 2014

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

SEVENTH FRAMEWORK PROGRAMME
Theme 3
Information and Communication Technologies

Deliverable D1.2

Control plane extension – Status of the SFA deployment

Grant Agreement number: **287581**

Project acronym: **OpenLab**

Project title: **OpenLab: Extending FIRE testbeds and tools**

Funding Scheme: **Large scale integrating project**

Project website address: **www.ict-openlab.eu**

Date of preparation: **August 29th, 2013**

Project co-funded by the European Commission within the Seventh Framework Programme (2007-2013)		
Dissemination Level		
PU	Public	**X**
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	

Document properties

People	
Document Editor:	Thierry Parmentelat (INRIA)
Authors:	Jordan Augé (UPMC) Loïc Baron (UPMC) Amine Larabi (INRIA) Nikos Mouratidis (CSE) Harris Niavis (UTH) Mohammed-Yasin Rahman (UPMC) Thierry Rakotoarivelo (NICTA) Florian Schreiner (FOKUS) Donatos Stavropoulos (UTH) Christos Tranoris (UoP) Alexander Willner (TUB)
History	
This version :	**1.0 (August 29th, 2013): Final**
Tagged releases :	1.0 (August 29th, 2013): Final 0.9 (July 28th, 2013): Prerelease 0.8 (June 17th, 2013): Draft, Mostly complete 0.1 (May 14nd, 2013): Skeleton
Git repo :	`https://git.ict-openlab.eu/?p=deliverable1.2.git`

Abstract

This document describes the progress made within Work Package 1 "Control Plane Extensions" over the second year of the OpenLab project. In a nutshell, it highlights our progress and achievements along the four strategical directions that we have been pursuing as part of our roadmap, and namely

SFA SfaWrap v3 now comes with support for the AM API v3, that exposes a more elaborate lifecycle for slices provisioning; for that reason SfaWrap expects a slightly different interface to be fulfilled by testbed drivers, although a compatibility layer is available until all drivers have migrated; we also describe in more details how the actual SFA interfaces have been implemented for both the NITOS and OSIMS testbeds.

Teagle The teagle codebase has taken advantage of the move towards supporting SFA to undertake a rather deep redesign, in order to achieve a fully recursive federation model; in addition we describe the implications for the UoP and TSSG testbeds.

FRCP As announced in the year-1 review meeting, we have made great progress towards specifying and implementing FRCP as a new esperanto for Experimental Plane tools to be able to manage resources; this paradigm, although under active development, is expected to have shortly support in user tools (EC, NEPI) as well as in testbeds (PLE).

MySlice A great deal of resources have been dedicated to the design and implementation of MySlice, both on the backend and frontend aspects. On the backend side, a generic software component named *manifold* has been isolated and packaged separately. On the frontend side, we have the seeds of a portal that can support a great portion of the experiment lifecycle.

Contents

1 Introduction **8**

2 Task 1.1 "Improve and Contribute to the SFA" **8**
2.1 Aggregate Manager API v3 . . . 8
2.1.1 Implementation . . . 8
2.1.2 Experiment lifecycle within AM API v3 . . . 10
2.2 Resource reservation . . . 11
2.3 Exposing NITOS testbed through SFA . . . 11
2.4 SFA interface for the OSIMS testbed . . . 15
2.4.1 Enabling OSIMS with SFA . . . 15
2.4.2 Advertised RSpec . . . 17

3 Task 1.2 - "Federation Framework Interoperability" **19**
3.1 Contributions to the SFA-Teagle interoperability . . . 19
3.1.1 SFA-related extensions of the Teagle framework . . . 20
3.1.2 Enabling PTM with SFA and SFAAdapter . . . 26
3.1.3 Integrating the TSSG IMS testbed with the control framework . . . 29
3.2 Contributions to the SFA-OMF interoperability . . . 30

4 Task 1.3 "Bridging SFA and the Experimental Plane" **31**
4.1 FRCP Design and Specification . . . 31
4.1.1 Messaging System, Naming and Addressing . . . 31
4.1.2 Protocol and Interactions . . . 32
4.1.3 Message Syntax . . . 33
4.2 FRCP Implementation and Deployment Status . . . 35
4.2.1 Implementation . . . 35
4.2.2 Deployment . . . 35

5 Task 1.4 "Extension of the Facility and Ops-oriented Tools" **36**
5.1 MySlice developments . . . 36
5.1.1 Description . . . 36
5.1.2 Evolutions to the MySlice web frontend . . . 36
5.1.3 MySlice portal functionalities . . . 37
5.1.4 Improvements to the backend . . . 38
5.2 Provisioning reservable resources through MySlice . . . 40
5.2.1 Developing NITOS Scheduler's plugin . . . 40
5.2.2 NITOS Scheduler's plugin layout . . . 40
5.3 Extension of the TSSG testbed facilities with OpenFlow . . . 42

6 Task 1.5 "Enlarging the facility Consortium" **43**

7 Conclusion **43**

List of Figures

1 Sliver allocation states . . . 9
2 SfaWrap on top of NITOS testbed . . . 12
3 NITOS and PlanetLab Europe federation via SfaWrap . . . 14
4 OSIMS testbed resources . . . 16
5 Teagle Architecture, Components and Layers . . . 20
6 Initial Approach for an SFA-enabled Teagle . . . 21
7 FITeagle Architecture, Components, and Layers . . . 22
8 FITeagle Overall Architecture and Components . . . 23
9 FITeagle Sequence Diagram (first steps only) . . . 25
10 PTM enhanced architecture with SFAAdapter . . . 26
11 TSSG IMS Testbed Integration . . . 29
12 Basic protocol interaction using FRCP. . . . 32
13 Screenshots from the MySlice portal . . . 37
14 MySlice dashboard . . . 38
15 NITOS Scheduler plugin layout . . . 40
16 Updating the slice . . . 41
17 TSSG Floodlight OpenFlow Controller . . . 42

List of Tables

1 AM API methods in v2 and v3 . . . 9

Listings

1 An rspec fragment for a lease . . . 11
2 Structure of NITOS RSpec. . . . 13
3 OSIMS SFA configuration. . . . 15
4 OSIMS advertised rspec. . . . 17
5 OSIMS lease . . . 18
6 OSIMS lease rspec example . . . 18
7 The SFAresp tag . . . 26
8 A simplified response RSpec . . . 27
9 How to set a user and password . . . 28
10 FRCP Message Format . . . 33
11 Example of a FRCP Message Exchange . . . 34

Acronyms

AM Aggregate Manager . . . 8
AMQP Advanced Message Queuing Protocol . . . 23
BonFIRE Building Service Testbeds on FIRE [10] . . . 21
CORBA Common Object Request Broker Architecture . . . 24
DSL Domain Specific Language . . . 19
FCI Federation Computing Interface [18] . . . 23
FI-PPP Future Internet Public Private Partnership [9] . . . 24
FI-WARE Future Internet Core Platform [8] . . . 24
FIRE Future Internet Research and Experimentation . . . 21
FRCP Federated Resource Control Protocol . . . 21
FSDL Federation Scenario Description Language [19] . . . 19
FSToolkit Federation Scenario Toolkit [19] . . . 19
GE Generic Enabler . . . 24
GENI Global Environment for Networking Innovations . . . 8
GID Global Identifier . . . 14
GUI Graphical User Interface . . . 23
GW Teagle Gateway . . . 21
IMS IP Multimedia Subsystem . . . 20
J2EE Java Platform Enterprise Edition . . . 25
JMS Java Message Service . . . 25
JSON JavaScript Object Notation . . . 24
LDAP Lightweight Directory Access Protocol . . . 24
OCCI Open Cloud Computing Interface . . . 21
OMF cOntrol and Management Framework . . . 23
OSGI Open Services Gateway initiative . . . 26
PII PII [20, 24]
PLE PlanetLab Europe . . . 14
PTM PanLab Testbed Manager . . . 19
RA Resource Adapter . . . 19
REST Representational State Transfer . . . 23
RSpec Resource Specification . . . 10
SFA Slice-based Federation Architecture [13] . . . 8

SFI SFA Command-Line Interface 23
SLA Service-Level Agreement 24
SMTP Simple Mail Transfer Protocol 24
SOAP Simple Object Access Protocol 23
SSH Secure Shell 24
Teagle Teagle [18, 22, 23, 25]
TSSG Telecommunications Software & Systems Group 20
URN Uniform Resource Name 11
VCT Virtual Consumer Testbed 19
VCTTool Virtual Customer Testbed Tool [22] 19
XML-RPC XML Remote Procedure Call 23
XML eXtensible Markup Language 24
XMPP eXtensible Messaging and Presence Protocol 23

1 Introduction

This document describes the progress made by the team involved in Work Package 1 Control plane extension over the first year of the project, i.e. from September 2011 to August 2012. It is organised along the various tasks that are defined in Work Package 1 as per the Project's "Description of Work" [3].

2 Task 1.1 "Improve and Contribute to the SFA"

During this second year project period, our contribution to Slice-based Federation Architecture was focused on extending our implementation of Slice-based Federation Architecture [13] (SFA), known as *Generic SFA Wrapper* (a.k.a. *SfaWrap*), by implementing the Aggregate Manager API v3 [17], adding support for time-based resource reservation, and enlarging the federation by exposing NITOS [12] and OSIMS testbeds through SFA.

2.1 Aggregate Manager API v3

The Aggregate Manager (AM) API, which is part of the Global Environment for Networking Innovations (GENI) [7] specifications, is the control plane interface through which experimenters discover resources and allocate resources to their slices in the form of slivers at resource providers. The AM API does not include resource specific interactions, application level interactions, or monitoring and management functions, which are deemed as belonging to the Experimental Plane.

2.1.1 Implementation

Following our contribution to the specification of the AM API v3, that was reported in the previous Deliverable D1.1, we have focused on coming up with a separate implementation of *SfaWrap* that supports AM API v3[1].

Firstly, we started by implementing the AM API v3 methods, knowing that since the AM API v2 [16], several changes have been introduced in terms of method names, signatures and scopes. Multiple methods have been renamed by removing the *sliver* term from the method names. The AM API v2 *CreateSliver* method has been broken into three new methods: *Allocate* to reserve the resources, *Provision* to instantiate the resources and *PerformOperationalAction* to start, stop or restart the resources. Also, *ListResources* is no longer used to list the resources that are attached to a slice, the new method *Describe* has been introduced to handle this functionality. Table 1 depicts the AM API methods changes between v2 and v3.

Secondly, our AM API v3 implementation had to manage a new allocation mechanism which specifies three sliver allocation states, and the transitions between them that are made through new methods: *Allocate*, *Provision* and *Delete* as depicted in Figure 1.

Then, we worked on the testbed's driver side of *SfaWrap*, by implementing an AM API

[1] SfaWrap v3 `http://git.onelab.eu/?p=sfa.git;a=shortlog;h=refs/heads/geni-v3`.

AM API v2	AM API v3
GetVersion	GetVersion
ListResources () ListResources (slice)	ListResources () Describe (slice)
CreateSliver	Allocate Provision PerformOperationalAction
DeleteSliver	Delete
SliverStatus	Status
RenewSliver	Renew
Shutdown	Shutdown

Table 1: AM API methods in v2 and v3

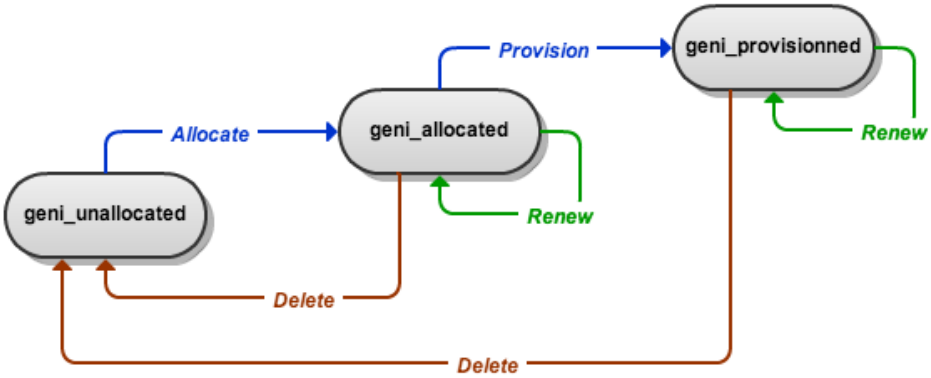


Figure 1: Sliver allocation states

v3 compliant driver for *PlanetLab*[2] and also the driver around the *Dummy*[3] testbed, which represents a key developer brick for testbed providers willing to write a driver for their own testbed.

Finally, we implemented a *v2 to v3 adapter*[4] which represents the glue between the already existing AM API v2-compliant drivers, such as: *NITOS*, *SensLab* [15], *Federica* [5], and the AM API v3 compliant interfaces of *SfaWrap*. The *v2 to v3 adapter* provides v3 compatibility to already existing v2-based testbed drivers, until their authors find the time to adapt their driver for a native support of AM API v3 if they want to take full advantage of the new lifecycle. It is hopefully a temporary component, since we expect all known drivers to migrate to this new model eventually.

[2]PlanetLab driver `http://git.onelab.eu/?p=sfa.git;a=tree;f=sfa/planetlab;hb=refs/heads/geni-v3`.
[3]Dummy driver `http://git.onelab.eu/?p=sfa.git;a=tree;f=sfa/dummy;hb=refs/heads/geni-v3`.
[4]v2-to-v3 adapter `http://git.onelab.eu/?p=sfa.git;a=blob;f=sfa/managers/v2_to_v3_adapter.py;hb=refs/heads/geni-v3`.

2.1.2 Experiment lifecycle within AM API v3

The experiment lifecycle, also called the experiment workflow process, is commonly broken down into the following steps [11]:

1. ***(CP)*** Account management and authentication
2. ***(CP)*** Resource discovery
3. ***(CP)*** Resource reservation
4. ***(EP)*** Experiment configuration
5. ***(EP)*** Experiment running
6. ***(EP)*** Data collection
7. ***(CP)*** Resource release

Steps 1, 2, 3 and 7 are referred to as the control plane, while steps 4, 5 and 6 correspond to the experimental plane. Being a control plane component for browsing, reserving and releasing resources offered by the federated networking testbeds, SfaWrap thus only addresses steps 1, 2, 3 and 7.

In this sense, the experimenter, using his client tool, starts by authenticating through the Registry API and retrieves his Certificate and Credentials granting him rights to reserve resources.

Then, through the AM API v3, the experimenter will discover resources by calling *ListResources*, which returns a detailed listing of the resources, known as an advertisement Resource Specification (RSpec).

Once the experimenter has selected the resources he wants and how to configure them, he produces a request RSpec that reflects his wishes, and reserves the wanted resources by calling *Allocate*, which takes the already produced request RSpec and his slice credentials as parameters. The aggregate then attempts to satisfy the experimenter's resource request by reserving the requested resources.

At that point, the resources have not been provisioned yet, giving the experimenter a chance to verify the reservation by calling *Status*, or check for corresponding resource availability at another aggregate. If it is acceptable, the experimenter calls *Provision* to set up the resources, which returns a manifest RSpec, listing the resources as reserved and initially configured for the experimenter. The purpose of returning a manifest is for cases when the actual reservation does not exactly fit the request, like e.g. if 1Gb/s of bandwidth is requested but due to policies, hardware and current usage, only 500Mb/s have been set aside for that slice.

Then, the experimenter will typically call *PerformOperationalAction* to start the resources (e.g. boot a machine).

Now that the resources are ready and started, the experimenter can configure and run his experiment, and then collect results and measurements. The experimenter can repeat this process as many times as needed, as far as the resources are still reserved to his slice, nowing that a slice expiration period can be extended via the AM API call *Renew*.

Finally, when the experimenter is done using the resources, he calls *Delete* to end his reservation. The aggregate then stops and clears all related resources. If a user fails to do so – which unfortunately happens a lot in real life –, the slice expiration period comes in, that is to say that *Delete* gets called automatically by that time.

2.2 Resource reservation

Within *SfaWrap*, resource discovery and reservation is made through RSpecs, in which we distinguish two types of resources: *Shared Resources* and *Exclusive Resources*. Reserving a shared resource (like the majority of PlanetLab Europe Nodes) is simply done by adding the selected resource description to Request RSpec. Reserving an exclusive resource (like NITOS nodes and channels, or SensLab nodes) needs more information regarding the time dimension of the reservation, which led us to introduce the concept of *Leases* in the RSpecs.

A *lease* is defined by a *slice_id* representing the Uniform Resource Name (URN) of the slice, a *start_time* representing the starting time of the time-based reservation, a *duration* representing the duration of the time-based reservation in terms of timeslots, and finally, the list of reserved exclusive resources referred by their *component_id*, which represents the URN of the resource. Below is an example of a lease that reserves NITOS exclusive resources (Nodes and Channels):

Listing 1: An rspec fragment for a lease

```
<lease slice_id="urn:publicid:IDN+omf:nitos+slice+plenitos" start_time="1363032000" duration="6">
  <node component_id="urn:publicid:IDN+omf:nitos+node+node001"/>
  <node component_id="urn:publicid:IDN+omf:nitos+node+node002"/>
  <channel channel_num="2"/>
  <channel channel_num="1"/>
</lease>
```

Leases are used in Request RSpec to reserve exclusive resources and are also present in the Advertisment RSpec, in a blacklist approach, in order to advertise about the already booked timeslots of exclusive resources.

2.3 Exposing NITOS testbed through SFA

Over the project's second year, we have worked, in collaboration with *University of Thessaly* (UTH) [21], on exposing *NITOS* testbed resources through SFA by developing a *SfaWrap* driver for *NITOS* testbed. Please note that the work described in this section was done during early this second year, and was based on OMF-v5; later during the project we have pursued the same goal but based on OMF-v6, and that work is described in section 3.2.

Figure 2 depicts the overall architecture of *SfaWrap* on top of *NITOS* testbed.

NITOS is a wireless experimental testbed which consists of nodes based on commercial WiFi cards and Linux Open Source drivers, distributed over several floors on a facility building. The control and management of the testbed is done using *OMF* combined with *NITOS Scheduler*, that enforces exclusive resource reservation, supports the concept of Slice and implements a spectrum slicing scheme preventing experiments to interfere with each other. *NITOS* testbed presents two types of exclusive resources: *Wireless Nodes* and *Wireless Channels*.

The development effort to wrap *NITOS* testbed using *SfaWrap* was divided as follows:

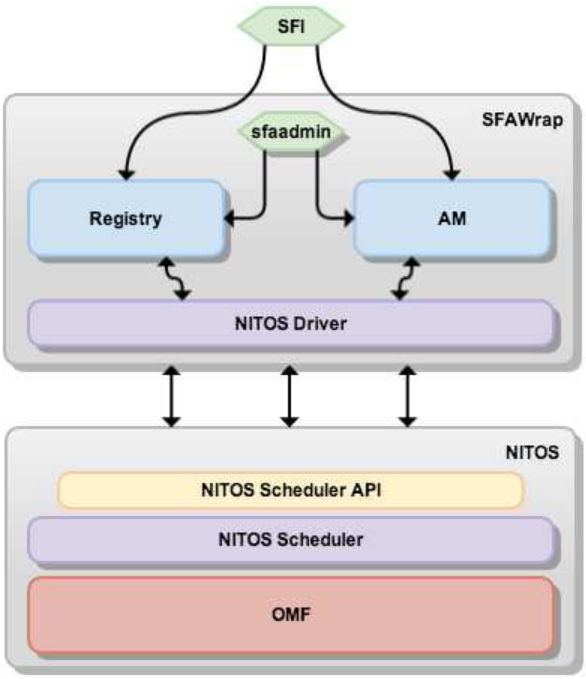


Figure 2: SfaWrap on top of NITOS testbed

NITOS Scheduler API. *SfaWrap*, being a control plane component, needs to communicate with *NITOS Scheduler* in order to manage NITOS resources, slices and users. Thus, UTH has developed the *NITOS Scheduler API* which is a XML-RPC interface that wraps the *NITOS Scheduler* with a set of methods for managing users and slices, and also for describing, reserving, provisioning and releasing resources.

NITOS Driver for SfaWrap. On the *SfaWrap* side, we had to develop a driver for *NITOS*[5] implementing the logic to deal with *NITOS* testbed specifities, and translating the Registry API calls and the AM API calls into *NITOS Scheduler API* calls. The *NITOS* driver performs on the one hand, Registry operations in order to keep synchronized the users and slices information between both *SfaWrap Registry* and *NITOS Scheduler*, and on the other hand, it describes, reserves and provisions *NITOS* resources.

NITOS RSpecs. In *NITOS* testbed, we distinguished two caracteristics of the resources. Firsty, all the resources are exclusive, and secondly, two types of resources are available: Nodes and Channels. Those two caracteristics lead us to define a new version of RSpecs for SfaWrap (aka. NITOSv1), in order to accurately describe NITOS resources.

As already mentioned in Section 2.2, we have taken advantage of Leases to express time-based reservation of resources within the RSpecs.

[5]NITOS v2-compliant driver `http://git.onelab.eu/?p=sfa.git;a=tree;f=sfa/nitos;hb=HEAD`.

On the other hand, we defined two new RSpec elements, namely: *spectrum* and *channel*, to describe and reserve wireless channels through RSpecs. For describing nodes, the node RSpec element was already existing in SfaWrap RSpecs.

The structure of NITOS RSpec is depicted in Listing 2.

```
<RSpec type="SFA" expires="2013-06-13T10:14:43Z" generated="2013-06-13T09:14:43Z">
  <statistics call="ListResources">
    <aggregate status="success" name="omf" elapsed="1.28529906273"/>
  </statistics>
  <network name="omf">
    <node component_manager_id="urn:publicid:IDN+omf+authority+cm" component_id="urn:publicid:IDN+omf:nitos
        +node+node001" component_name="node001" site_id="urn:publicid:IDN+omf:nitos+authority+sa">
      <hostname>node001</hostname>
      <location country="unknown" longitude="39.360839" latitude="22.949989"/>
      <position_3d x="1" y="1" z="5"/>
      <exclusive>TRUE</exclusive>
      <gateway>nitlab.inf.uth.gr</gateway>
      <granularity>1800</granularity>
      <hardware_type>orbit</hardware_type>
    </node>
    <node component_manager_id="urn:publicid:IDN+omf+authority+cm" component_id="urn:publicid:IDN+omf:nitos
        +node+node002" component_name="node002" site_id="urn:publicid:IDN+omf:nitos+authority+sa">
      <hostname>node002</hostname>
      <location country="unknown" longitude="39.360839" latitude="22.949989"/>
      <position_3d x="1" y="2" z="6"/>
      <exclusive>TRUE</exclusive>
      <gateway>nitlab.inf.uth.gr</gateway>
      <granularity>1800</granularity>
      <hardware_type>orbit</hardware_type>
    </node>
    <spectrum>
      <channel channel_num="36" frequency="5180.0" standard="IEEE802_11a"/>
      <!-- ... -->
      <!-- other channel descriptions go here -->
      <!-- ... -->
      <channel channel_num="13" frequency="2472.0" standard="IEEE802_11b_g"/>
    </spectrum>
    <lease slice_id="urn:publicid:IDN+omf:nitos+slice+nbarati" start_time="1349796600" duration="320">
      <node component_id="urn:publicid:IDN+omf:nitos+node+node001"/>
      <node component_id="urn:publicid:IDN+omf:nitos+node+node002"/>
    </lease>
    <lease slice_id="urn:publicid:IDN+omf:nitos+slice+plenitos" start_time="1363032000" duration="6">
      <node component_id="urn:publicid:IDN+omf:nitos+node+node001"/>
      <node component_id="urn:publicid:IDN+omf:nitos+node+node002"/>
      <channel channel_num="2"/>
      <channel channel_num="1"/>
    </lease>
  </network>
</RSpec>
```

Listing 2: Structure of NITOS RSpec.

NITOS Driver deployment. Finally, we federated *NITOS* testbed with *PlanetLab Europe* via SfaWrap, in order to allow the execution of cross-testbed experiments that combine wired and wireless nodes. A trust relationship is established between NITOS and PlanetLab Europe (PLE) by exchanging their respective Global Identifier (GID)'s. As a result, NITOS users can reserve and provision PLE resources and vice-versa. The overall architecture of the federation between NITOS and PLE is depicted in Figure 3.

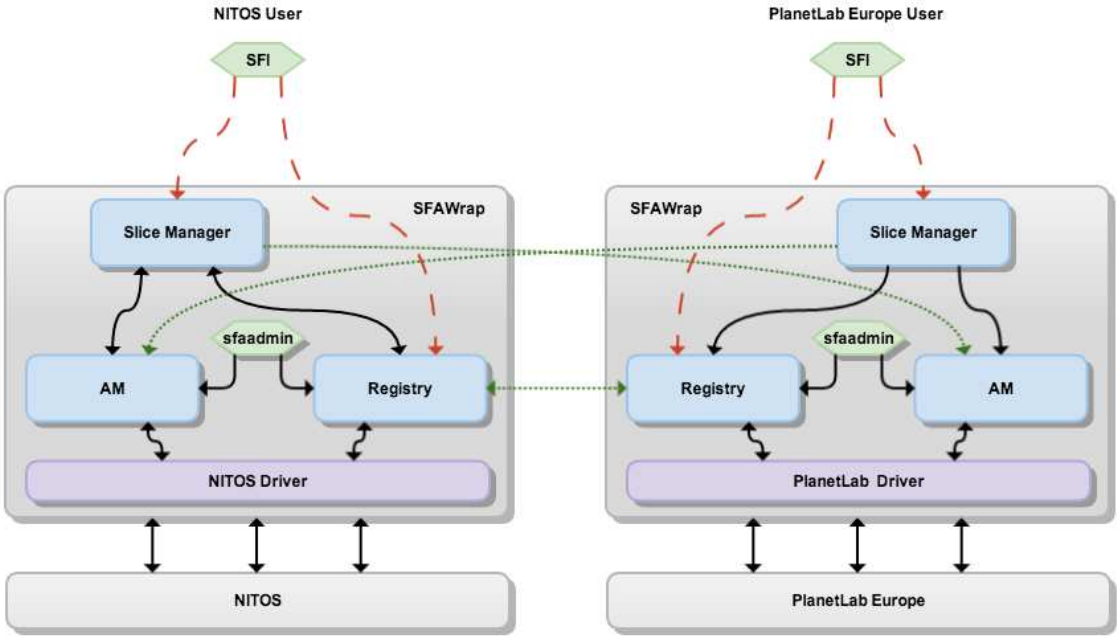


Figure 3: NITOS and PlanetLab Europe federation via SfaWrap

2.4 SFA interface for the OSIMS testbed

For the latest developments in the OSIMS testbed of University of Patras, we implemented and deployed an SFA solution, in order to make the testbed resources available for reservation to the rest of the federation and any SFA compliant tools. The next sections describe our solution to be SFA compliant through details of the RSpec exposed.

2.4.1 Enabling OSIMS with SFA

Figure 4 displays how SFA Aggregate Manager and registry are deployed, with the rest of our resources. The SFA AM are hosted on a Linux Ubuntu machine. The implementation is based on python scripts while the configuration follows concepts of the SFA service configuration.

Our typical configuration is as follows:

```
========== Category = SFA_REGISTRY
SFA_REGISTRY_ROOT_AUTH = upatras
========== Category = SFA_DB
SFA_DB_PASSWORD = sfadbpwd
========== Category = SFA_ UOP
SFA_UOP_URL = http://nam.ece.upatras.gr/fedway/sfa/xmlrpc.php
========== Category = SFA
SFA_INTERFACE_HRN = upatras
SFA_GENERIC_FLAVOUR = p2e
SFA_API_LOGLEVEL = 2
```

Listing 3: OSIMS SFA configuration.

Our implementation was based on a clone of the SFA driver originally written for Federica, which was implementing the communication with an xmlrpc server. Therefore we have created a P2EDriver class inside the SFA service deployment. This class implements several commands for responding to external events (like list resources, manage the life-cycle of slivers, etc). The P2EDriver class makes certain calls to the class P2EShell which is responsible of communicating with our XMLRPC implementation on the host machine where other services of the testbed are located.

In detail, the XMLRPC module is implemented by PHP scripts that are in front of our fedway[6] component. Fedway is a small gateway that monitors and logs all requests towards our testbed. The XMLRPC calls are transformed into requests that our testbed understands and pushed via the fedway to an FCI enabled layer (a TeagleGateway implementation) down to our domain manager (the PTM).

[6] `http://nam.ece.upatras.gr/fedway/`

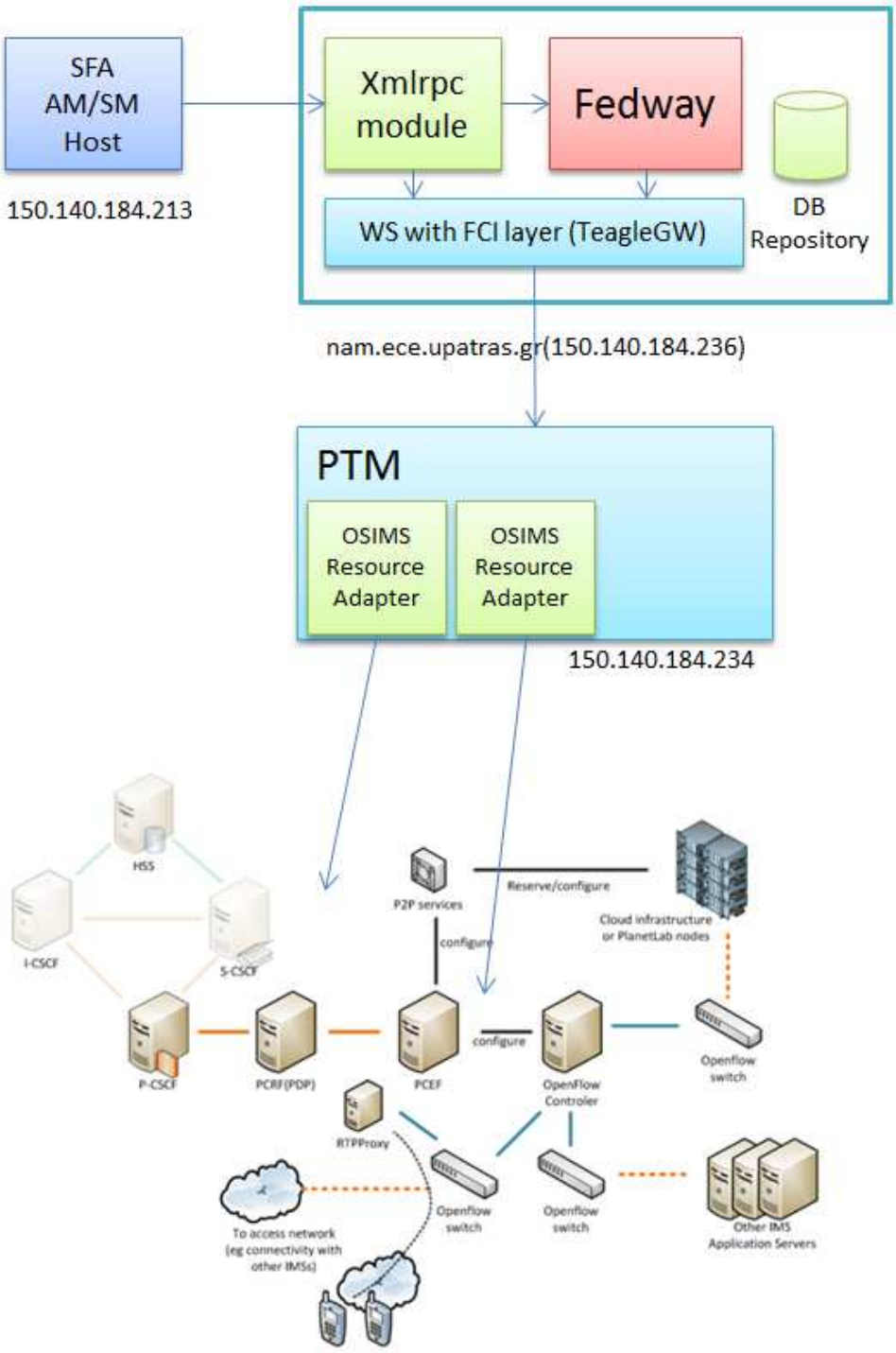


Figure 4: OSIMS testbed resources

2.4.2 Advertised RSpec

Our OSIMS testbed as of today advertises the following (part) RSpec on listing resources:

```
<rspec type="advertisement" valid_until="2013-05-20T16:03:57+03:00" generated="2013-05-20T15:03:57+03:00">
 <statistics call ="ListResources">
  <aggregate status="success" name="upatras" elapsed="0.1"/>
 </statistics >
 <network name="upatras">
  <node component_manager_id="urn:publicid:IDN+upatras+authority+cm" component_id= "urn:publicid:IDN+
       upatras:p2e+node+orOSIMSCreateSubscriberAccount" component_name="imscreateuseraccount" site_id="urn:
       publicid:IDN+upatras:p2e+authority+sa">
   <displayname>imscreateuseraccount</displayname>
   <servicename>imscreateuseraccount</servicename>
   <description>The resource for creating a subscriber account on OSIMS</description>
   <taxonomy>OSIMS</taxonomy>
   <location country="unknown" longitude="21.7885" latitude="38.2845"/>
   <settings>
    <setting name="Username" description="username used for OSIMS subscriber"> </setting>
    <setting name="Password" description=""></setting>
    </settings>
   <lease from="2013-05-20T15:03:57+03:00" until="2013-05-21T15:03:57+03:00">false</lease>
  </node>
< node component_manager_id="urn:publicid:IDN+upatras+authority+cm" component_id="urn:publicid:IDN+upatras:
      p2e+node+orOSIMSCreateFullAccess" component_name="imsfullaccess" site_id="urn:publicid:IDN+upatras:p2e+
      authority+sa">
  <displayname>imsfullaccess</displayname>
  <servicename>imsfullaccess</servicename>
  <description>Resource for creating an SSH account and providing shell access to all servers of OSIMS</description
       >
  <taxonomy>OSIMS</taxonomy>
  <location country="unknown" longitude="21.7885" latitude="38.2845"/>
  <settings>
  <setting name="SSHUsername" description="username used for OSIMS subscriber"></setting>
  <setting name="SSHPassword" description="password used for OSIMS subscriber"></setting>
  <setting name="CoreReset" description="if true recreates the OSIMS IMSCore server and resets it to the initial state.
       All user modifications are lost ."> </setting>
   <setting name="OpenSIPSReset" description="if true recreates the OSIMS OpenSIPS server and resets it to the
       initial state. All user modifications are lost ."></setting>
   <setting name="MediaServerReset" description="if true recreates the OSIMS MediaServer server and resets it to the
       initial state. All user modifications are lost ."> </setting>
  </settings>
 <lease from="2013-05-20T15:03:57+03:00" until="2013-05-21T15:03:57+03:00">false</lease>
 </node>
 <node component_manager_id="urn:publicid:IDN+upatras+authority+cm" component_id="urn:publicid:IDN+upatras:
      p2e+node+orOSIMSFloodlightAccess" component_name="imsfloodlightaccess" site_id="urn:publicid:IDN+upatras
      :p2e+authority+sa">
  <displayname>imsfloodlightaccess</displayname>
  <servicename>imsfloodlightaccess</servicename>
  <description>Enable OSIMS OpenFlow controller Floodlight</description>
  <taxonomy>OSIMS</taxonomy>
  <location country="unknown" longitude="21.7885" latitude="38.2845"/>
  <settings>
  </settings>
 <lease from="2013-05-20T15:03:57+03:00" until="2013-05-21T15:03:57+03:00">false</lease>
  </node>
  <node component_manager_id="urn:publicid:IDN+upatras+authority+cm" component_id="urn:publicid:IDN+upatras
       :p2e+node+orOSIMSOpenFlowSwitchAccess" component_name="imsopenflowaccess" site_id="urn:publicid:IDN+
       upatras:p2e+authority+sa">
  <displayname>imsopenflowaccess</displayname>
  <servicename>imsopenflowaccess</servicename>
  <description>Enable access to OSIMS openflow switch</description>
```

```
  <taxonomy>OSIMS</taxonomy>
  <location country="unknown" longitude="21.7885" latitude="38.2845"/>
  <settings>
  </settings>
  <lease from="2013-05-20T15:03:57+03:00" until="2013-05-21T15:03:57+03:00">false</lease>
  </node>

<!-- MORE Nodes follow -->

 </network>
</rspec>
```

Listing 4: OSIMS advertised rspec.

Our RSpec has similarities with most published RSPecs of other testbeds. However there are some extra elements that may help tools to display information to experimenters. For example there are description elements, a potential taxonomy and display name of the resource.

Within the RSpec there is a lease element. It has a slot of 1 day. That is the next available day that this resource is available. Tools that consume this rspec and want to reserve resources, just need to turn into true the lease element. For example, if an experimenter via a tool wants to reserve the OpenFlow switch of OSIMS on the corresponding lease element one should write:

```
<lease from="2013-05-20T15:03:57+03:00" until="2013-05-21T15:03:57+03:00">true</lease>
```

Listing 5: OSIMS lease

This means that the node will be reserved for one day from 20/5/2013 15:00 UTC for the next 24 hours.

Of course together with the lease, the requester should configure the requested resource. As an example, assume that we want to reserve the IMS core testbed and get full access to the core for that particular period. The following request can be prepared like in listing 6:

```
<rspec type="request" >
 <network name="upatras">
< node component_manager_id="urn:publicid:IDN+upatras+authority+cm" component_id="urn:publicid:IDN+upatras:
      p2e+node+orOSIMSCreateFullAccess" component_name="imsfullaccess" site_id="urn:publicid:IDN+upatras:p2e+
      authority+sa">
  <servicename>imsfullaccess</servicename>
  <settings>
  <setting name="SSHUsername">ctranoris</setting>
  <setting name="SSHPassword">mypass</setting>
  <setting name="CoreReset">true</setting>
   <setting name="OpenSIPSReset">true</setting>
   <setting name="MediaServerReset">false</setting>
  </settings>
 <lease from="2013-05-20T15:03:57+03:00" until="2013-05-21T15:03:57+03:00">true</lease>
 </node>
</network>
</rspec>
```

Listing 6: OSIMS lease rspec example

3 Task 1.2 - "Federation Framework Interoperability"

The aim of Task 1.2 – "Federation Framework Interoperability" is to make emerged federation frameworks interoperable and come up with a recursive federation model and implementation that allows for federation at any granularity. In the subsequent sections the latest developments are described that were conducted after last Deliverable 1.1 was issued. It was agreed within the OpenLab consortium, that the SFA acts as the common denominator for resource federation frameworks. Therefore, this task focuses on the interoperability efforts between existing frameworks and SFA.

3.1 Contributions to the SFA-Teagle interoperability

Figure 5 depicts the Teagle [18, 22, 23, 25] federation architecture and its components, as were developed in PII [20, 24]. The overview is divided into four different layers as follows:

User Level. On this level the experimenters and testbed owners have different tools to interact with the Teagle framework. A testbed owner can register and manage his testbed and involved resources using a web portal. These resources are then shown in the Virtual Customer Testbed Tool [22] (VCTTool). The VCTTool allows the user the select, configure and interconnect arbitrary resources from involved testbeds. This set of resources are then described, stored and forwarded as a Virtual Consumer Testbed (VCT). Based on this VCT the Federation Scenario Toolkit [19] (FSToolkit) can be used to conduct an experiment. The FSToolkit is a tool used for defining experiments in a textual way and enables federation scenarios by accessing different testbeds via different authentication methods and API schemes. It supports a textual Domain Specific Language (DSL) called Federation Scenario Description Language [19] (FSDL). Extensions made in FSToolkit are described in Section 3.1.2.

Federation Level. The Teagle architecture follows a centralized approach. A single Teagle instance is used to enforce the legal federation framework on a technical level by carrying out according policies. This central systems is composed of different components to evaluate and authorize incoming requests, to store the requested VCTs, to resolve resource dependencies and to execute the needed requests to the involved testbeds in order. Enhancements on the Teagle architecture, that lead to the development of the FITeagle framework, is described in Section 3.1.1.

Testbed Control Level. In order to join a Teagle federation, a testbed owner must run a PanLab Testbed Manager (PTM). This PTM is responsible to receive resource provisioning requests and to call the according Resource Adapters (RAs). The RAs are then translating the requests to resource specific commands to control the certain resources of a testbed. Enhancements on the PTM level are discussed in Section 3.1.1.

Testbed Level. The testbeds contain the actual resources. Since the Teagle federation architecture allows everything to be a resource, a wide-ranging diversity of testbeds can be sup-

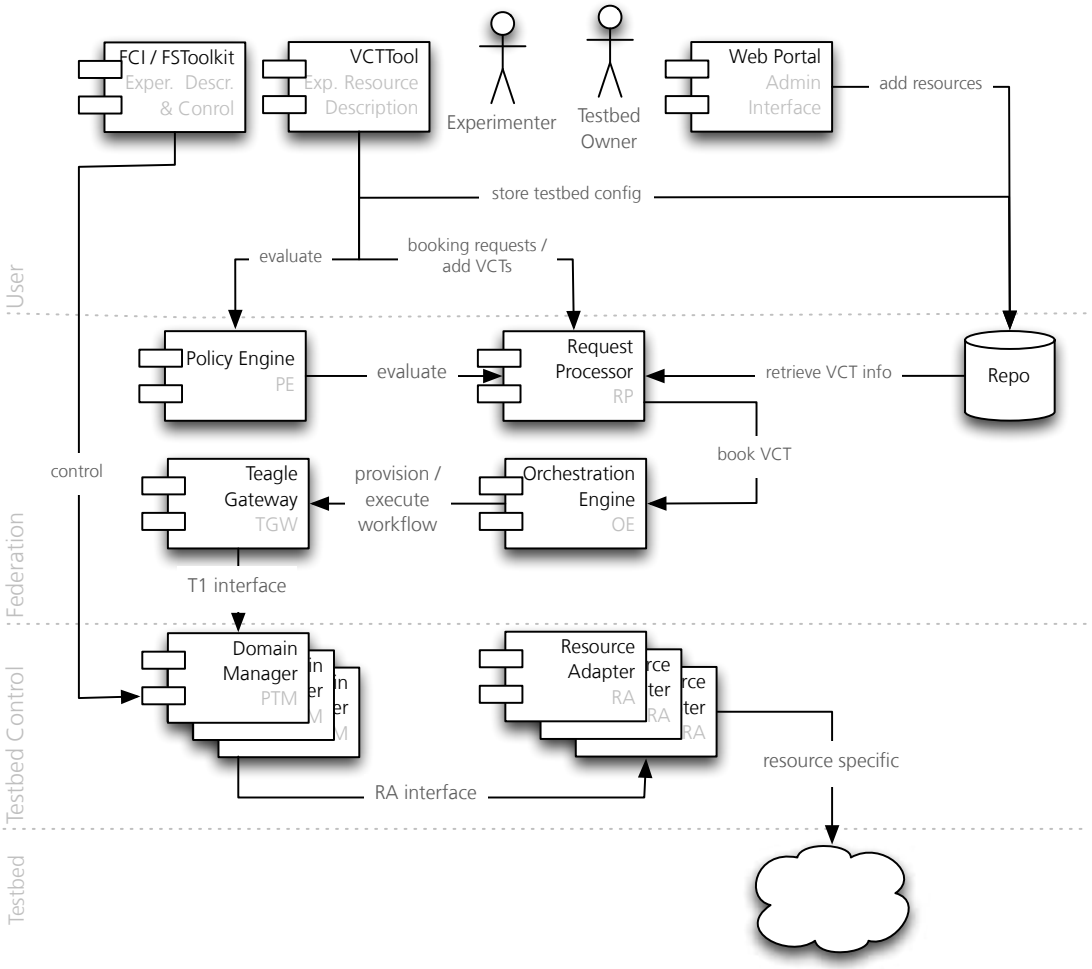


Figure 5: Teagle Architecture, Components and Layers

ported. In particular the enhancements of the IP Multimedia Subsystem (IMS) testbed from the Telecommunications Software & Systems Group (TSSG) are described in Section 3.1.3.

3.1.1 SFA-related extensions of the Teagle framework

The Teagle [18,22,23,25] federation architecture as defined within the PII [20,24] project, exposes four different interfaces (cf. Figure 5):

1. **Policy Engine Interface.** Used for example by the VCTTool in order to indicate which resources are allowed to be connected with each other.

2. **Request Processor Interface.** Used to provision, start, and stop a given VCT.

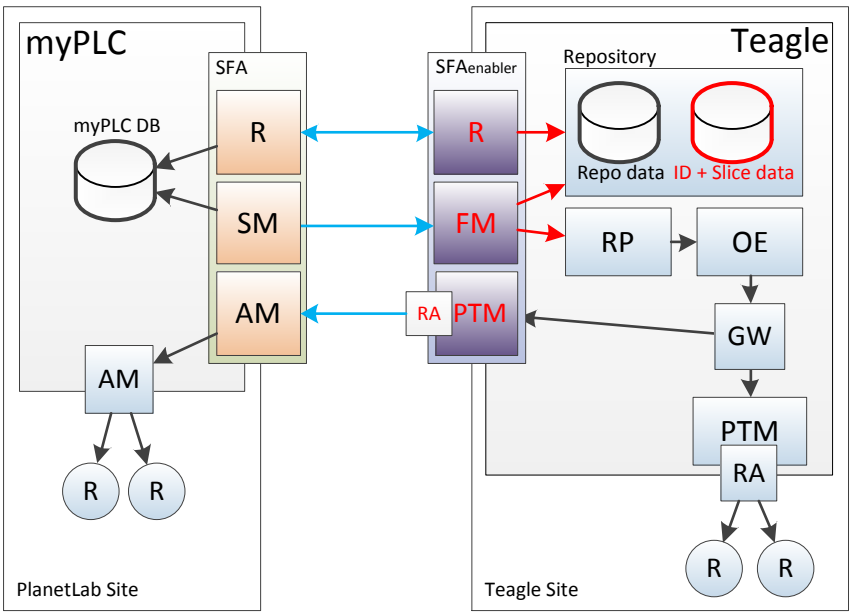


Figure 6: Initial Approach for an SFA-enabled Teagle

3. **Repository Interface.** Used by the by portal or the VCTTool to create, read, update, delete VCTs, resource types or testbeds.

4. **Domain Manager Interface.** Each involved testbed must implement this interface that is used by the Teagle Gateway (GW).

The initial mapping between these interfaces and the SFA interfaces was described in Deliverable 1.1 and is reminded in Figure 6.

In line with the overarching goal of this task, namely "Federation Framework Interoperability", other federation mechanisms must be taken into account. It turned out that within the Future Internet Research and Experimentation (FIRE) community, other interfaces than SFA are also considered. For example within the Building Service Testbeds on FIRE [10] (BonFIRE) project an extended version of the Open Cloud Computing Interface (OCCI) is being developed. But mainly, the Federated Resource Control Protocol (FRCP) (cf. Section 4) is getting more attention as an complementary interface, as it addresses Experimental Plane.

Therefore, it was decided to follow a more sustainable approach by redesigning the Teagle architecture to be agnostic to the actual federation protocol. This new design is being implemented within the FITeagle[7] framework with a current focus on SFA AM v3 [17] and ProtoGeni Registry v1 [14] interoperability. In Figure 7 a high-level architecture is shown, that is based on the aforementioned Teagle architectural layers. The basic concepts of testbeds, control components on testbed level, a federation layer, and user clients on top, were inherited.

[7] `http://fiteagle.org`

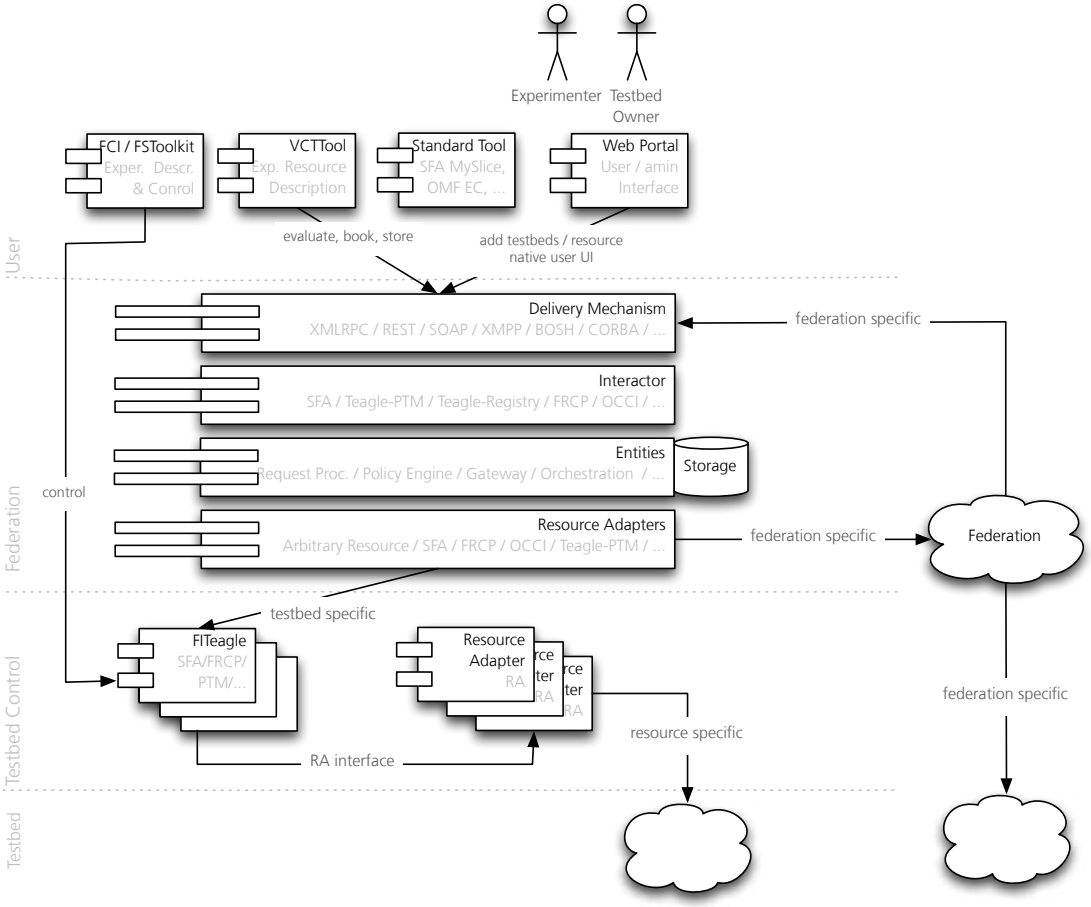


Figure 7: FITeagle Architecture, Components, and Layers

The most crucial design changes were: the partitioning of the core functionalities into uncoupled modules; the abstraction of the offered interfaces (Delivery Mechanisms) and protocols (Interactors); the refactoring of core functionalities into protocol-independent libraries (Entities); and the possibility for testbed owners to offer resources by different federation mechanisms using a single tool.

In Figure 8 the overall architecture with its components is illustrated: it follows a modular event-driven design pattern, which is going to be described in the subsequent paragraphs and will be subject of subsequent publications. This architecture allows the framework to scale with the number of handled messages, to develop separate components following a test-driven approach, and to be very agnostic regarding external interfaces.

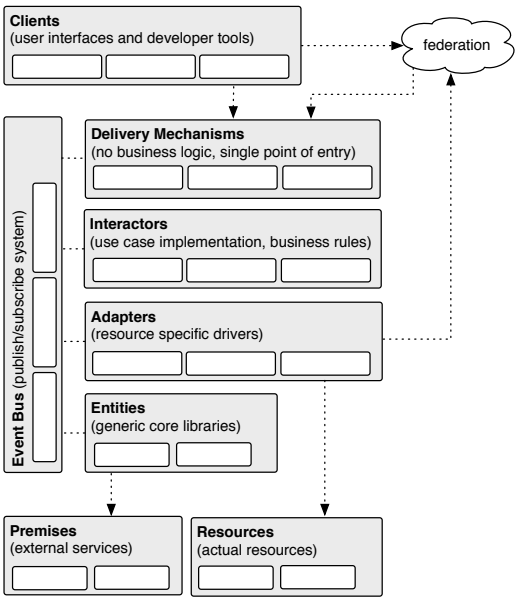


Figure 8: FITeagle Overall Architecture and Components

Clients As described in Section 3.1 there are several existing clients for federated resource provisioning and control. They are well established in their own user communities and some are under active development. Clients range from command line tools, e.g. SFA Command-Line Interface (SFI), to more sophisticated graphical user interfaces, e.g. MySlice or VCTTool, as well as developer kits, e.g. Federation Computing Interface [18] (FCI). Depending on the experimenter's working environment, specific tools will be used and it is unlikely that a user would change to another one. Therefore, these clients are out of scope of the FITeagle development, but are used to ensure compatibility with other implementations. The main target is not to offer new tools, but to support these clients and to use them for acceptance testing. The current focus lies on SFA compliant systems and will then be extended to FRCP and Teagle related clients. Furthermore, a web-based Graphical User Interface (GUI) is in development.

Delivery Mechanisms Analog to the number of clients, there are also countably many standardized interfaces that are used by federation and experiment control protocols. Each federation mechanism uses one or more different web service interfaces to communicate with the users, tools or federated testbeds. While the SFA exposes information using XML Remote Procedure Call (XML-RPC), the cOntrol and Management Framework (OMF) offers two interfaces based on eXtensible Messaging and Presence Protocol (XMPP) and Advanced Message Queuing Protocol (AMQP) respectively. Additionally, the Teagle framework uses Simple Object Access Protocol (SOAP) and Representational State Transfer (REST) protocols. Therefore, the `Delivery Mechanisms` module, which is the single point of entry to FITeagle, offers several interfaces at once and handles the transport layer security. Currently, this module contains an SSL-secured XML-RPC interface to support SFA and will be extended to support REST, eX-

tensible Messaging and Presence Protocol (XMPP), AMQP and SOAP for other mechanisms. On this level no business logic is implemented, and a message contents is only forwarded to the according protocol implementation in the `Interactors` module, and for this reason adding other delivery mechanisms is a very lightweight task. Furthermore, other legacy interfaces such as Common Object Request Broker Architecture (CORBA) or other proprietary message protocols could also be exposed on this level, in order to interact with a wide range of services. This approach also allows to offer the same protocol using different interfaces simultaneously, and decouples the involved marshalling medium like eXtensible Markup Language (XML) or JavaScript Object Notation (JSON) from the underlying protocol.

Interactors In this module, the federation and experiment control protocol-specific logic is implemented in separate packages. It is agnostic to the used delivery mechanism. The package receives the messages from the associated `Delivery Mechanisms` module as an object and acts accordingly. In order to perform the needed actions, the central idea is that mainly protocol-agnostic libraries from the `Entities` and resources from the `Adapters` modules are going to be used. As a result, each interactor will contain only a few logic rules, and the core libraries are reused by all interactors, since they might share many functionalities.

Entities This module contains all the core libraries that are needed by more than one interactor. They either implement the offered functionalities their self or delegate them to another existing service in the `Premises` module. For example a persistence library implementations can be used either in-memory storage for testing, sqlite storage for simple persistence, or existing SQL compatible storage services. Other possible functionalities include scheduling, orchestration, authentication, authorization or even Service-Level Agreement (SLA) evaluation mechanisms. These are in the process of being ported from the Teagle framework.

Premises This module represents existing services that are available at the related testbed. These services might need to be consulted by a core library. An example is a user database functionality that delegates the request to an existing Lightweight Directory Access Protocol (LDAP) database. Other examples are the use of several Future Internet Core Platform [8] (FI-WARE) Generic Enablers (GEs) in the Future Internet Public Private Partnership [9] (FI-PPP) context, or the integration into a local payment system in a commercial environment.

Adapters The `Adapters` module contains the resource drivers, also named RAs. The concept of adapters is mainly adopted from the Teagle architecture and gets extended. Each RA encapsulates one or multiple resource instances of a single resource type in the `Resources` module by offering a unified interface for resource description, provisioning, control, monitoring, and release. The translation from the incoming function call to the actual outgoing call is very resource-specific. It can be mapped for example to Telnet or Secure Shell (SSH) connections, Simple Mail Transfer Protocol (SMTP) commands or even could invoke a call to another federation. Many existing Teagle RAs will be adopted and on this level also another federation is handled as a group of available resources.

Resources Like in the Teagle architecture, everything is a resource and they can contain or be linked with other resources. Examples are: a coffee machine, physical or virtual machines, the network or specific network components, software packages, services, testbeds or other federations. In this context also a testbed itself or another federation is a resource.

Event Bus The core to loosely interconnecting the different modules, and to exchanging messages and events is an asynchronous, bi-directional publish-subscribe system. This allows to potentially distribute several components and therefore scale the framework as demands raises or to add and remove components without interfering with the running system. Depending on the environment, different implementations are going to be supported. In an Java Platform Enterprise Edition (J2EE) environment Java Message Service (JMS) compatible system could be used. Other options are for example XMPP or AMQP.

The current implementation for the FITeagle framework is being tested by several SFA compliant user tools such as: sfi, omni, jFed, and MySlice. An UML Sequence Diagram of the message flow is depicted in Figure 9.

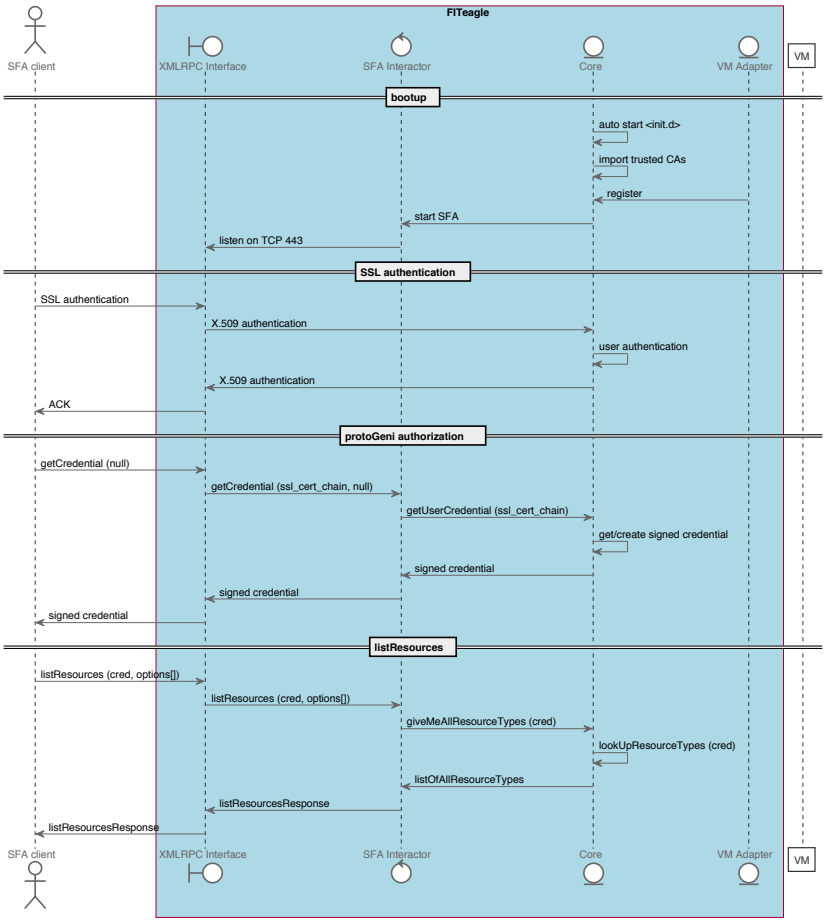


Figure 9: FITeagle Sequence Diagram (first steps only)

3.1.2 Enabling PTM with SFA and SFAAdapter

PTM is the domain manager installed in a testbed in order to expose its resources via an interface. The PTM interface is a secure SOAP interface, accessible publicly by applications and usually via the Teagle Gateway (The interface is also known as the T1 interface). PTM is based on Open Services Gateway initiative (OSGI) (http://www.osgi.org/Main/HomePage) and consists of several loaded bundles, like the Resource Adaptation Layer and the Resource Adapters. Each Resource Adapter is responsible for wrapping a resource API to a homogeneous API, responding to RESTful commands (i.e. POST,GET, PUT, DELETE) on a resource instance. Having these in mind we have created inside the PTM a Java bundle called SFA Adapter. The SFA adapter enumerates all the Resource Adapters which reside in the PTM and exposes them via an RSpec. It is assumed that such a call comes via the RESTful interface of the TeagleGW. The same result can be returned if tools implement the SOAP calls of the PTM (towards T1 interface).

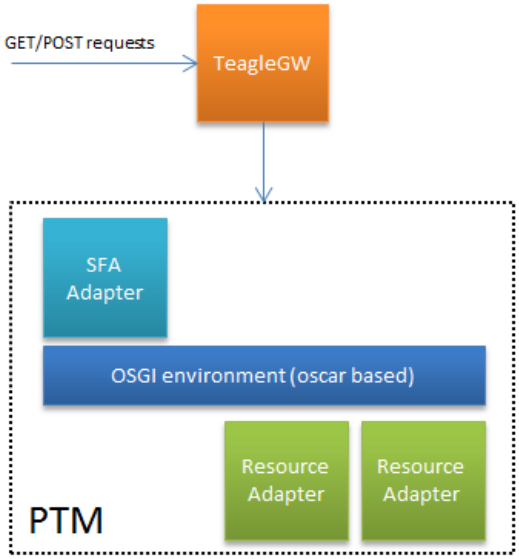


Figure 10: PTM enhanced architecture with SFAAdapter

To be compliant with other tools the RSpec is wrapped around the TeagleGW response, in the SFAresp tag, like the following code shows:

```
<?xml version="1.0" encoding="utf-8"?>
<sfaadapter>
<uuid type="string">uop.sfaadapter-14</uuid>
<SFAReq>null</SFAReq>
<SFAresp>RSPEC encoded</SFAresp>
</sfaadapter>
```

Listing 7: The SFAresp tag

The following is a part of the RSpec response of the PTM. The example is from the PTM installed in the UoP testbed:

```
<rspec xmlns="http://www.protogeni.net/resources/rspec/2" type="advertisement"
 valid_until=" 2013-05-23T17:12Z " generated="2013-05-23T16:12Z">
<statistics call="ListResources">
<aggregate status="success" name="upatras" elapsed="0.1"/>
</statistics>
<network name="upatras">
<node component_manager_id="urn:publicid:IDN+upatras+authority+cm"
component_id="urn:publicid:IDN+upatras:teaglegw+node+sflow" component_name="sflow"
site_id="urn:publicid:IDN+upatras:p2e+authority+sa">
<location country="GR" longitude="21.7885" latitude="38.2845"/>
<settings>
<setting name="target" ></setting>
<setting name="sflow_id" ></setting>
</settings>
</node>
<node component_manager_id="urn:publicid:IDN+upatras+authority+cm"
component_id="urn:publicid:IDN+upatras:teaglegw+node+qos_rate_limiting" component_name="qos_rate_limiting"
site_id="urn:publicid:IDN+upatras:p2e+authority+sa">
<location country="GR" longitude="21.7885" latitude="38.2845"/>
<settings>
<setting name="vif" ></setting>
<setting name="ingress_policing_rate" ></setting>
<setting name="ingress_policing_burst" ></setting>
</settings>
</node>
<node component_manager_id="urn:publicid:IDN+upatras+authority+cm"
component_id="urn:publicid:IDN+upatras:teaglegw+node+imscreateuseraccount"
component_name="imscreateuseraccount"
site_id="urn:publicid:IDN+upatras:p2e+authority+sa">
<location country="GR" longitude="21.7885" latitude="38.2845"/>
<settings>
<setting name="Username" ></setting>
<setting name="Password" ></setting>
</settings>
</node>
<node component_manager_id="urn:publicid:IDN+upatras+authority+cm"
component_id="urn:publicid:IDN+upatras:teaglegw+node+uop_ovswitch001" component_name="uop_ovswitch001"
site_id="urn:publicid:IDN+upatras:p2e+authority+sa">
<location country="GR" longitude="21.7885" latitude="38.2845"/>
<settings>
<setting name="IP" ></setting>
<setting name="attachedVifs" ></setting>
</settings>
</node>
<node component_manager_id="urn:publicid:IDN+upatras+authority+cm"
component_id="urn:publicid:IDN+upatras:teaglegw+node+vm_ofattached" component_name="vm_ofattached"
site_id="urn:publicid:IDN+upatras:p2e+authority+sa">
<location country="GR" longitude="21.7885" latitude="38.2845"/>
<settings>
<setting name="InstalledOS" ></setting>
<setting name="VIFs" ></setting>
<setting name="ElasticIP" ></setting>
</settings>
</node>
</network>
</rspec>
```

Listing 8: A simplified response RSpec

The rspec returned by a PTM is simplified. It contains information found by browsing the Resource Adapters installed within the PTM. This is done by using OSGI services that queries all installed bundled for an implementation of the Java RA interface. If they bundle responds, the SFAAdapter browses for each available configurable properties. So each node in the rspec corresponds to a Resource Adapter, the component name is the Resource Adapter name, and the settings are the configurable properties available. This is the response when requesting the rspec.

To create a resource adapter (i.e. to configure a resource) an rspec should be injected with the requested settings of that particular resource. For example, to request a new username and password towards the IMS, the following RSpec should be posted towards the teagle gateway:

```
<rspec type="request">
<network name="upatras">

<node component_manager_id="urn:publicid:IDN+upatras+authority+cm"
component_id="urn:publicid:IDN+upatras:teaglegw+node+imscreateuseraccount"
component_name="imscreateuseraccount" site_id="urn:publicid:IDN+upatras:p2e+authority+sa">
<location country="GR" longitude="21.7885" latitude="38.2845"/>
<settings>
<setting name="Username" >myusername</setting>
<setting name="Password" >mypwd</setting>
</settings>
</network>
</node>
```

Listing 9: How to set a user and password

To identify and expose via the rspec all the deployed Resource Adapters within the PTM, the BundleContext of the OSGI container is used to get access to the service registry. A BundleContext is a bundle's execution context within the OSGI container. The context is used to grant access to other methods so that this bundle can interact with the OSGI container. BundleContext methods allow a bundle to:

- Subscribe to events published by the Framework.
- Register service objects with the Framework service registry.
- Retrieve ServiceReferences from the Framework service registry.
- Get and release service objects for a referenced service.
- Install new bundles in the Framework.
- Get the list of bundles installed in the Framework.
- Get the Bundle object for a bundle.
- Create File objects for files in a persistent storage area provided for the bundle by the OSGI container.

The BundleContext object is created for the SFAAdapter bundle when the bundle is started. The Bundle object associated with a BundleContext object is called the context bundle. The

BundleContext object is passed to the BundleActivator.start(BundleContext) method during activation of the SFAAdapter context bundle. The same BundleContext object is passed to the BundleActivator.stop(BundleContext) method when the SFAAdapter context bundle is stopped. The SFAAdapter BundleContext object is only valid during the execution of its context bundle; that is, during the period from when the context bundle is in the STARTING, STOPPING, and ACTIVE bundle states. Since all Resource Adapters within PTM are bundles, the SFAAdapter uses the getServiceReferences method of the BundleContext to get a collection of ServiceReference objects registered within the OSIGI container implementing the RA.class service.

3.1.3 Integrating the TSSG IMS testbed with the control framework

The TSSG efforts to make our IMS testbed available through the broader federation of testbeds has progressed according to plan. As per the analysis in the earlier stages of the project we deemed it important to fulfil our goal of integrating our testbed with the Teagle framework. In conjunction with our partners in UoP, the PTM has been installed and configured on our testbed in the TSSG. A test Resource Adapter and resource instance were also installed and tested successfully and our PTM is now listed in the TeagleGW hosted in UoP. This is setting the foundation for exposing the resources offered by the TSSG testbed as available services to the community of interested experimenters.

(a) PTM hosted on TSSG VM

(b) TSSG PTM listed in UoP hosted TeagleGW

Figure 11: TSSG IMS Testbed Integration

A separate and interesting scenario for integrating the TSSG IMS testbed has arisen from the WONDER project – a successful proposal from the 2nd Open Call for experiments. The experimenters from Portugal Telecom and Deutsche Telekom propose to evaluate WebRTC service delivery mechanisms, including IMS, and inter-domain scenario testing is on their wish list. To this end the integration of the TSSG NGN testbed with UoPs OSIMS testbed has gained some traction over the past while and some early baselining and planning has taken place, namely at the new experiment negotiation meeting in Paris and related conference calls.

3.2 Contributions to the SFA-OMF interoperability

As described in section 2.3, our first deployment of SFA over OMF has been based on OMF-v5 in an effective , although a bit awkward way. We had announced during the review meeting in Madrid that we were intending to integrate more tightly SFA into OMF-v6, taking advantage of that redesign to make the coupling smoother and with a wider applicability.

The integration of SFA in OMF version 6 is a joint development between NICTA and UTH. During the first year of the project, NICTA developed an AM implementation prototype with an *SFA*-compatible interface, which would be adopted in the new version of OMF. Building on its experience in implementing the *SFAWrap* for the *NITOS* testbed, UTH contributed several enhancements to the *OMF SFA* implementation. More specifically, the efforts of UTH and NICTA were focused on integrating *NITOS Scheduler* functionalities in the *OMF SFA AM* implementation. We have extended the basic resource model to include new reservation information. Furthermore, we upgraded the AM API version from *GENI AM v1* to *GENI AM v2*. We will further continue upgrading the AM API as required (e.g. to v3), similar to the work done for the *SFAWrap*.

The final goal for this task is to deliver a native *SFA API* support for OMF testbeds, so as to ease the federation of the existing OMF testbeds and possible new ones. The *OMF SFA AM* will be provided as a new entity, which could be installed on top of existing OMFv6 deployments. Following the design approach of OMFv6 the new AM will be treated as another resource, responsible for controlling the testbed. Due to that fact, in addition to the SFA interface, *OMF AM* is also exposing an FRCP (Described in section 4) interface, through which users are able to control the underlying resources during the experiment.

The latest implementation of *OMF SFA*, together with information about its development, are available here: `https://github.com/mytestbed/omf_sfa`.

4 Task 1.3 "Bridging SFA and the Experimental Plane"

As introduced in the previous OpenLab Deliverable D1.1, we proposed to develop an open protocol named Federated Resource Control Protocol (FRCP), which will be used by various user/facility entities to communicate with testbed resources, and by testbed resources themselves to interact with each other. This section provides an update on our recent work in developing FRCP.

4.1 FRCP Design and Specification

We published the design and the specification of the FRCP protocol online earlier this year [6]. This subsection summarises that online document.

4.1.1 Messaging System, Naming and Addressing

As introduced in D1.1, we adopted a publish-and-subscribe (pubsub) message system for handling communication between resources and the entities interacting with them. In such a system, participants can create topics, subscribe to them and publish messages in them. A message which is published in a given topic is forwarded via the messaging system to all the subscribers of that topic. As such, the communication is effectively one-way. Any reply to a message can only be performed by another published message and some mutually agreed convention on how to relate messages. However, we do assume that the messaging system is reliable and will attempt to deliver every message in a timely manner. However, there is no guarantee that every message will be delivered to all subscribed entities. The primary reason for a failed delivery will be due to an overflowing inbox for a particular subscriber.

The basic message pattern assumes that every active resource is associated with a topic (address) and all messages to and from this resource are published to this topic. The messages *create*, *configure*, *request*, and *release* are sent to the resource, while the *inform* message originates from the resource itself. The *inform* message is usually in response to a received message, but can be sent on its own. For instance, a resource may periodically publish its state, while on the other hand it may respond to multiple requests with a single *inform*. However, to simplify client logic we adopt the convention that every message contains a message ID and that the *inform* message includes a list of message IDs it received since the last *inform* message.

Any entities and resources in this proposed scheme can subscribe to none or many topics on the pubsub system. Any pubsub system that supports the above can be used for this proposed messaging scheme, for example XMPP-based or AMQP-based pubsub systems.

A resource has a globally unique ID. This unique ID is selected by the parent of a resource upon its creation. An implementation of this proposed scheme must adopt a convention to map this globally unique ID to a unique topic name (aka topic address) within the deployed pubsub system. We propose the following mapping between a resource's globally unique ID and its unique topic name. Assuming that the resource ID is '123456', the communication layer used is XMPP-based pubsub, the resource access the pubsub system via the server 'foo.bar.com'; then, the mapped topic for that resource would be: xmpp://123456@foo.bar.com

A resource must subscribe to the topic address which corresponds to its globally unique name, and must publish any *inform* message that it issues to this topic address. In addition if that

inform message is a reply to a request message which contained a specific <replyto> field with a topic set to it, then the *inform* message must also be published to that reply topic. A resource may subscribe to any other topics as required by the user.

4.1.2 Protocol and Interactions

The basic protocol consists of a message being sent by a requester to a component (or resource). The component may accept the message and perform the requested associated actions. This may lead to further messages being sent to an observer. We introduced a separate observer to allow for different messaging frameworks. For instance, in an RTP like framework the observer is usually the requester itself, while in a publish-subscribe framework the observer is a stand-in for a publication.

The protocol consists of five messages *inform*, *configure*, *request*, *create*, and *release*. A basic interaction pattern is shown in Figure 12.

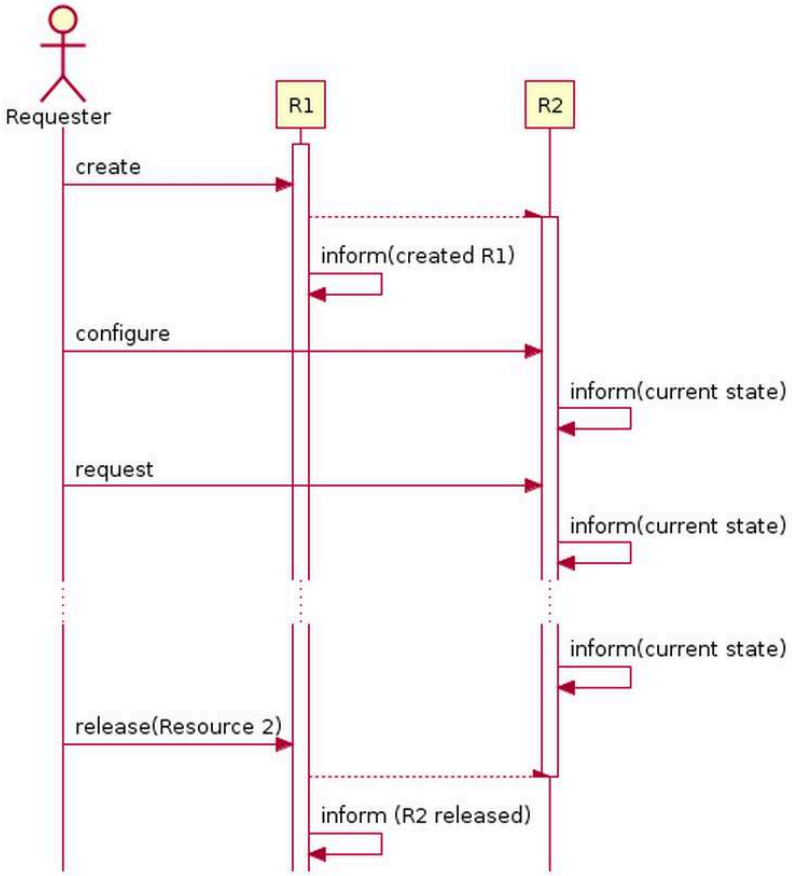


Figure 12: Basic protocol interaction using FRCP.

4.1.3 Message Syntax

FRCP messages can be marshalled using either XML or JSON. The remaining of this page will describe the XML formats of FRCP messages, the corresponding JSON formats are described on our JSON format for FRCP page. The schema definitions for XML messages of FRCP are available online[8].

The generic format of a message element is described in listing 10:

```
<MTYPE xmlns="http://schema.mytestbed.net/omf/X.Y/protocol" mid=ID>
  <src>RID</src>
  <ts>TIMESTAMP</ts>
  <replyto>TOPIC</replyto>
  ....
</MTYPE>
```

Listing 10: FRCP Message Format

Here is how to interpret this snippet :

- X.Y = the version of the protocol (currently 6.0)
- MTYPE = the type of message, either "create", "configure", "request", "inform", or "release"
- ID = a globally unique ID for this message
- RID = a globally unique ID for the resource that published this message. Note that every entities are resources. Thus a software acting on behalf of a user, such as an experiment controller, will also have an unique resource ID. We propose to use an URI for this ID.
- TIMESTAMP = the Posix/Unix time in second when this message was generated by its publisher. This 'ts' and the 'mid' child elements are used to prevent message replays.
- TOPIC = optional. If the publisher of this message would like any replying messages to be published to a specific topic address, then it must set TOPIC to that address in the 'replyto' element.

This message element may then have child elements, which further describe various information specific to the message type. The online FRCP specification describes each message type in details.

In addition to these message type-specific child elements, FRCP also defines two other child elements which are used in various message types. The first one is the 'guard' element, which carries constraint information. When this child element is present, an entity will only act on a received message if it satisfies the constraints described in the 'guard' element. This 'guard' element is described in more details at the end of this page. The second one is the 'props' element which declares various properties related to a message and which is described next.

[8] http://schema.mytestbed.net/omf

FRCP provides an optional mechanism to verify that a given message was published by a given entity. The online FRCP specification provides a detailed description of the adopted Authentication and Authorization mechanisms.

The following listing 11 shows an example of the exchange of a *configure* and an *inform* FRCP messages between two FRCP-enabled entities "node123" and "iperf456".

```
<configure xmlns="http://schema.mytestbed.net/omf/6.0/protocol" mid="83ty28">
  <src>xmpp://node123@domainA.com</src>
  <ts>1360895974</ts>
  <properties xmlns:iperf="http://foo.com/iperf">
     <iperf:target type='hash'>
       <ip type='string'>192.168.1.2</ip>
       <port type='fixnum'>5001</port>
     </iperf:target>
  </properties>
</configure>

<inform xmlns="http://omf.mytestbed.net/omf/6.0/protocol" mid="d934l8">
  <src>xmpp://iperf456@domainB.com</src>
  <ts>1360895982</ts>
  <cid>83ty28</cid>
  <itype>STATUS</itype>
  <properties xmlns:iperf="http://foo.com/iperf">
     <iperf:target type='hash'>
       <ip type='string'>192.168.1.2</ip>
       <port type='fixnum'>5001</port>
     </iperf:target>
  </properties>
</inform>
```

Listing 11: Example of a FRCP Message Exchange

4.2 FRCP Implementation and Deployment Status

4.2.1 Implementation

The FRCP protocol is currently implemented in the latest release of the OMF framework, which allows experimenters to describe, instrument and orchestrate experiments using resources across multiple testbed facilities. This OMF version 6 release is available online on the OMF website[9].

The source code of the OMF implementation of FRCP is also available as a reference for developers looking to interface their own experiment controller or testbed resources with other FRCP-enabled entities[10]. This reference implementation is in Ruby, but a third party may use any other programming language to implement FRCP.

Other partners in OpenLab have proposed to implement FRCP communication stacks for Experiment-Plane tools such as NEPI and TEAGLE. The progress on these tasks are available as part of other related OpenLab Deliverables.

4.2.2 Deployment

FRCP is currently used as part of OMF6 on NICTA's wireless indoor testbed. Other OpenLab partners are also in the process of testing OMF6 deployment on their own testbeds (e.g. UTH).

Furthermore OMF6 is also being deployed as an upgrade to the previously available OMF5-friendly slice option on PlanetLab Europe. NICTA and the PlanetLab Europe team at INRIA have discussed and agreed on the required deployment steps to insure authentication of FRCP entities on PlanetLab. They are currently working together in implementing the final steps of this roadmap, in essence the slice-side of PLE slivers is mostly finished as of MyPLC-5.2.6, and we expect to be able to announce the availability of this feature to the experimenter community shortly.

[9] http://mytestbed.net/doc/omf/file.INSTALLATION.html
[10] https://github.com/mytestbed/omf/tree/master/omf_common/lib/omf_common

5 Task 1.4 "Extension of the Facility and Ops-oriented Tools"

5.1 MySlice developments

5.1.1 Description

MySlice consists in a user-centric tool, tailored to support the full users' experiment lifecycle, from setup through completion. It includes a portal functionality allowing users to register, authenticate and request slices. A dashboard presents them with an overview of the federation, as well as the resources and slices they have access to. Finally, as it embeds an extended SFA client, MySlice allows users to browse and select resources of interest for their experiments. In particular, it features third-party measurement and monitoring information, as well as visualization and navigation components allowing users to make sense of available resources.

The challenge MySlice solves is that the different platforms of interest for an experimenter typically realize one given functionality (SFA deals with the control plane, OMF with the experimental plane, TopHat with measurements, etc.) which is decoupled from users needs at the various stages of the experiment lifecycle. Through MySlice, a user can get a consistent access to those different services. For instance, the tool takes advantage of measurements originating from TopHat and OML, and bring them into a context useful for a users (measurements about a testbeds, a resource, a slice, ...).

We distinguish two parts in MySlice, denoted backend and frontend, corresponding to the two major bricks in the architecture. For the sake of clarity, we have distinguished the *Manifold* component which is at the heart of the MySlice backend, and that provides an interconnection framework for heterogeneous data, as well as a set of users interfaces. MySlice inherits the architecture of this component as well as its extensions capabilities: gateways allows new platforms to be added, such as testbeds via SFA; plugins allow for specialization of the web GUI in order to provide a consistent and full-featured user interface, still adapted to the needs of each testbed.

Notable features of MySlice include: balancing uniformity and heterogeneity; combining multiple sources of data; hiding complexity from users; providing a variety of user interfaces; ensuring ease of development for developers. Full reference and documentation is available on the project website: `http://www.myslice.info` and the related wiki: `http://trac.myslice.info`. A journal paper describing MySlice (in addition to SFAWrap) is available in [1].

In this section, we focus on describing the major improvements that have been done in the component, towards the realization of an integrated portal for the federation of experimental facilities.

5.1.2 Evolutions to the MySlice web frontend

Until recently and for historical purposes, MySlice relied on the *Joomla!* PHP framework for the implementation of its web user interface. This framework was not judged production-ready and caused several issues for operating the component. This motivated the migration towards *django*, a famous python-based framework, in order to build a more robust, easily deployable and manageable user interface. Since it builds on the Python programming language, this moved allowed us to get a more consistent codebase with the backend with which it peers.

After an initial development phase that consisted in bringing functionalities existing in the

Joomla frontend in a production ready-state, we are now considering the adaptation of plugins to the new django framework. Despite the move to another programming language (from PHP to python), the amount of work to port existing plugins is minimal. Indeed, most of the functionalities are developed in Javascript, which is common to both architectures. The Python/PHP part consist only in a few lines of codes that serves for the initialization of the plugins and its dependencies. It was a design decision to keep the actual interface for plugins, in order not to confuse existing developers, and to plug existing work as seamlessly as possible in the new architecture.

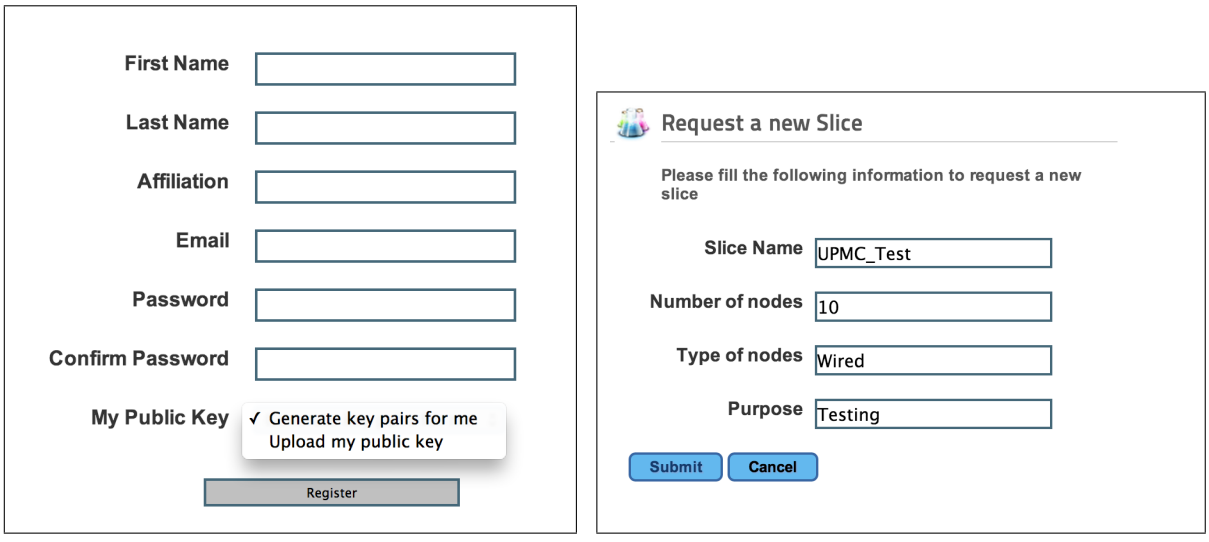
This last step will allow us to open a unified portal for real users before the end of 2013. This portal will have the two main functionalities exposed below:

- what we name the portal component, responsible for handling user registration and their slice requests, to be validated by a responsible PI,
- the slice manangement component, in charge of allowing users to browse and reserve resources for their slice.

5.1.3 MySlice portal functionalities

The first page that users will be faced with is the MySlice portal, allowing them to register and login, to create slices, and to browse and reserve testbed resources. For some of the more complex tasks, users will be guided through a series of simple steps. During registration for example, depending on their choices, they will be invited to generate a keypair that will be used for authentication to the testbeds, or simply upload their public key and delegate some of their rights to MySlice. More details about the portal functionality can be found in [2].

Figure 13 present two screenshots of the user registration and slice request pages.



(a) User Registration Form (b) Slice Request Form

Figure 13: Screenshots from the MySlice portal

Dashboard Once a user has been authenticated by MySlice and authorized by a testbed, he can access that testbed's resources. The MySlice dashboard (Fig. 14) is the page that welcomes users upon login. It provides an overview of the user's account and authorization credentials, the various types of resources that the user can access through the different federated authorities, as well as a list of the slices with which the user is associated. To generate this page, MySlice issues a series of queries to retrieve information about the different SFA objects related to the logged user, such as user, authority or slice information.

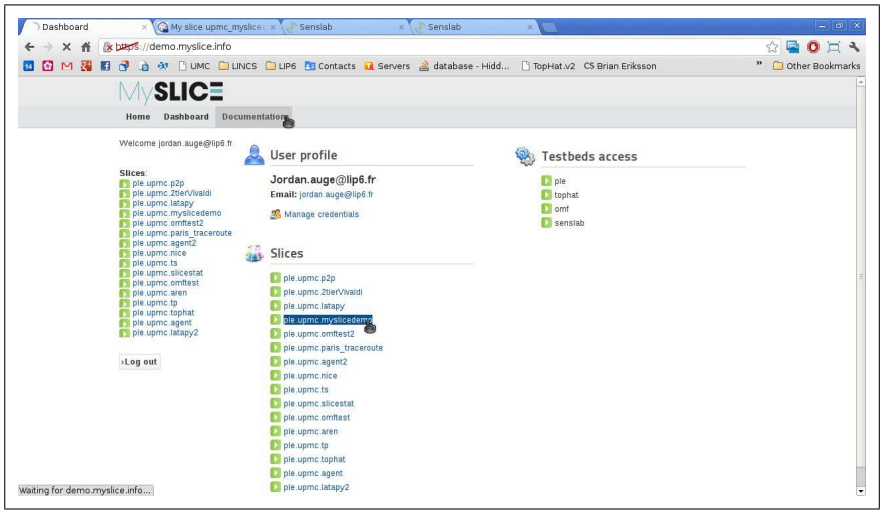


Figure 14: MySlice dashboard

Future work Future planned work towards release consists in adding functionalities for PI to get notified of new requests, and to be informed of pending users and slices to be validated.

5.1.4 Improvements to the backend

Improvements towards a production version As mentioned earlier, we have reworked the MySlice backend so as to isolate its generic core, that can be used in completely different contexts. This generic core has been named *manifold*, and represents a very substantial portion of the code, with all the specifics (SFA interface, DB interfaces, and so on) being implemented as *gateways* in the manifold framework.

MySlice relies heavily on its XMLRPC interface, both for communication with end users, and to support the queries originating from the frontend. The API has thus been extended to be a full-featured XMLRPC server. A set of useful management scripts have also been developed, that are necessary for the daily operation of the service. All these components now support a series of options that can be configured either with default options, command line arguments, or options in a configuration file. An initialization script, coupled with the configuration file, allows for a seamless usage of *manifold*.

Packages for major linux distributions (fedora, debina and ubuntu) are now provided for the stable version of manifold, allowing users to easily come up with an operational installation of the component. Nightly builds are also available for developers.

Multiple improvements have been done to the backend in order to stabilize a critical set of functionalities, required for a production-grade instance of MySlice, such as a robust handling of users queries, extensive logging and error reporting.

Finally, a shell allows a user to run commands to the federation, which turns out to be a particularly useful feature for quick access to data, or for troubleshooting some issues with either the component or the platforms.

Extension of the SFA gateway The SFA gateway allows SFA information (resources, slices, users and authorities) to be exposed to MySlice. To better fit the component architecture, and for performance issue, the SFA gateway has been rewritten to support multiple asynchronous requests within a single thread. This is especially useful in a context where several SFA gateways are defined in a MySlice deployment, so that all the SFA calls - each of which being likely to take quite some time - can be run in parallel rather than sequentially.

In addition, gateways are responsible for performing SFA actions on behalf of the user. For MySlice to be able to issue these calls, the preferred mode is for the user to delegate her rights to the tool (by means of a delegated SFA credential). However, this can be somehow complex for the user, and a second mode has been devised so that this complexity is fully hidden and taken over by MySlice. The SFA gateway now has a so-called "managed" mode that can boostrap and handle all necessary authentication and authorization tokens. Accountability is preserved by using the native SFA delegation feature, and these actions can be safely traced back to the originating user.

5.2 Provisioning reservable resources through MySlice

The last step towards the goal of making the NITOS testbed SFA-compliant, was the development of a plugin for MySlice, in order to provision NITOS reservable resources. To this end, UTH developed the NITOS Scheduler plugin which parses NITOS RSpecs and provides a decent layout for reserving resources.

5.2.1 Developing NITOS Scheduler's plugin

MySlice framework has a modular structure based on the concept of plugins for implementing different core functionalities, like "query editing", "data display" and "resource allocation". NITOS Scheduler plugin lies in the "data display" category, where resources that match the selected query are being displayed.

Like every MySlice plugin, the NITOS Scheduler plugin consists of a PHP[11] script, together with a CSS file which is responsible for rendering, and some javascript code which is responsible for the interactions with both the user and other plugins. We tried to build a structural plugin that will allow other testbeds to take advantage of it, in order to expose their reservable resources through SFA. As soon as a testbed uses the same RSpecs as NITOS, it is a matter of a few modifications in the code and its different types of reservable resources will appear in the NITOS Scheduler's plugin tabs.

5.2.2 NITOS Scheduler's plugin layout

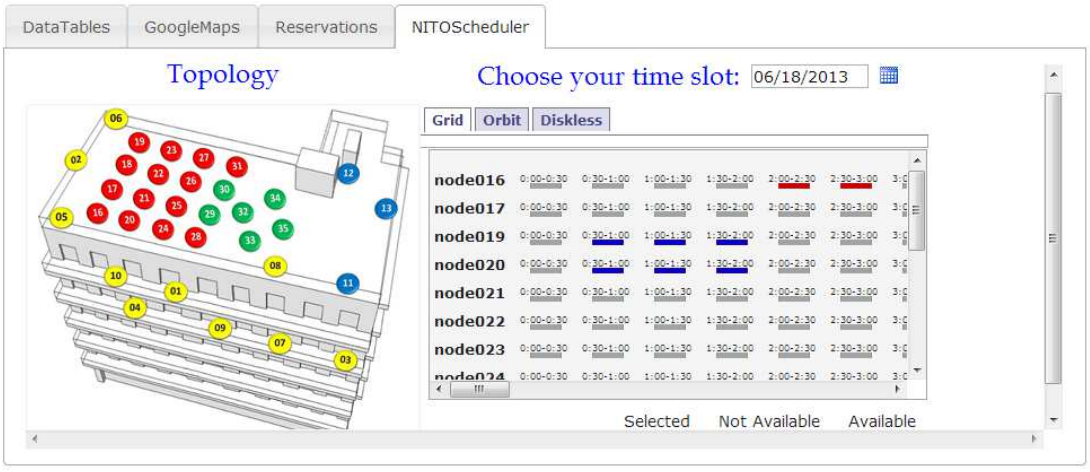


Figure 15: NITOS Scheduler plugin layout

A user can start the reservation procedure by filtering his query in the first section of plugins and based on his selection, our plugin will provision him the corresponding resources as depicted in figure 15. At the left section of the layout, a picture of the testbed's topology is displayed and at the right section lies a tab for each different type of resource. Subsequently, he will check

[11]This plugin is currently being ported to the new django-base MySlice framework.

their availability, and select those that match his requirements. Finally, he will see the selected resources in the "resource allocation" section of the MySlice portal as depicted in figure 16 and update his slice with the reservation information of the corresponding time slots.

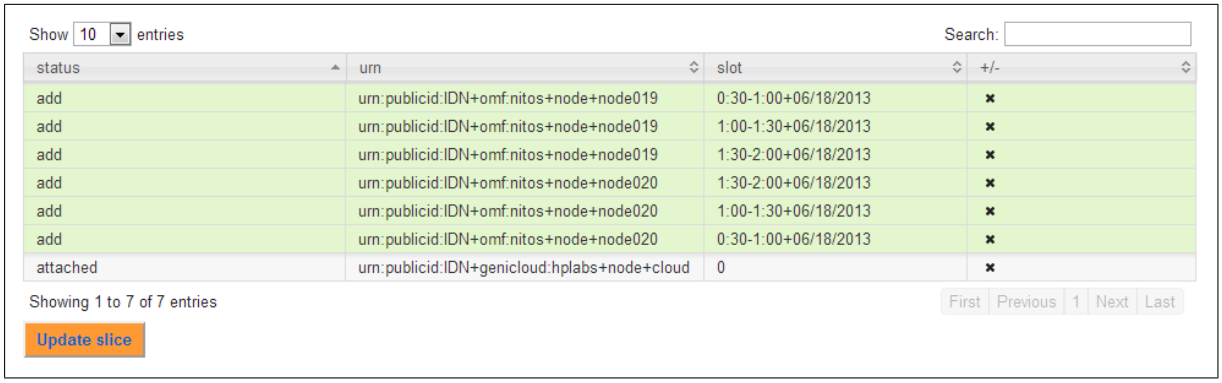


Figure 16: Updating the slice

5.3 Extension of the TSSG testbed facilities with OpenFlow

TSSGs contribution to extending the operation-oriented tools within the facility has focused on our implementation of an OSGI-based OpenFlow framework for experimentation. This has been implemented using the Floodlight Open SDN controller.

At the lower level of the system, the framework statically deploys elements such as Open-VSwitch, OpenFlow enabled switches and OpenFlow controller parts for controlling, provisioning and administering underlying network resources. An API implementation in Java/RabbitMQ at the upper layer permits access to the architectures control plane logic and in turn access to OpenFlow which manages the network elements.

A network provisioning engine has been created with two functions in mind. Firstly, to enable a logical network overview to be discovered composing of nodes, links and interfaces. Secondly, the provisioning of these resources is also being enabled with the ability to host intelligent logic from P2P routing algorithms in the overlay manager.

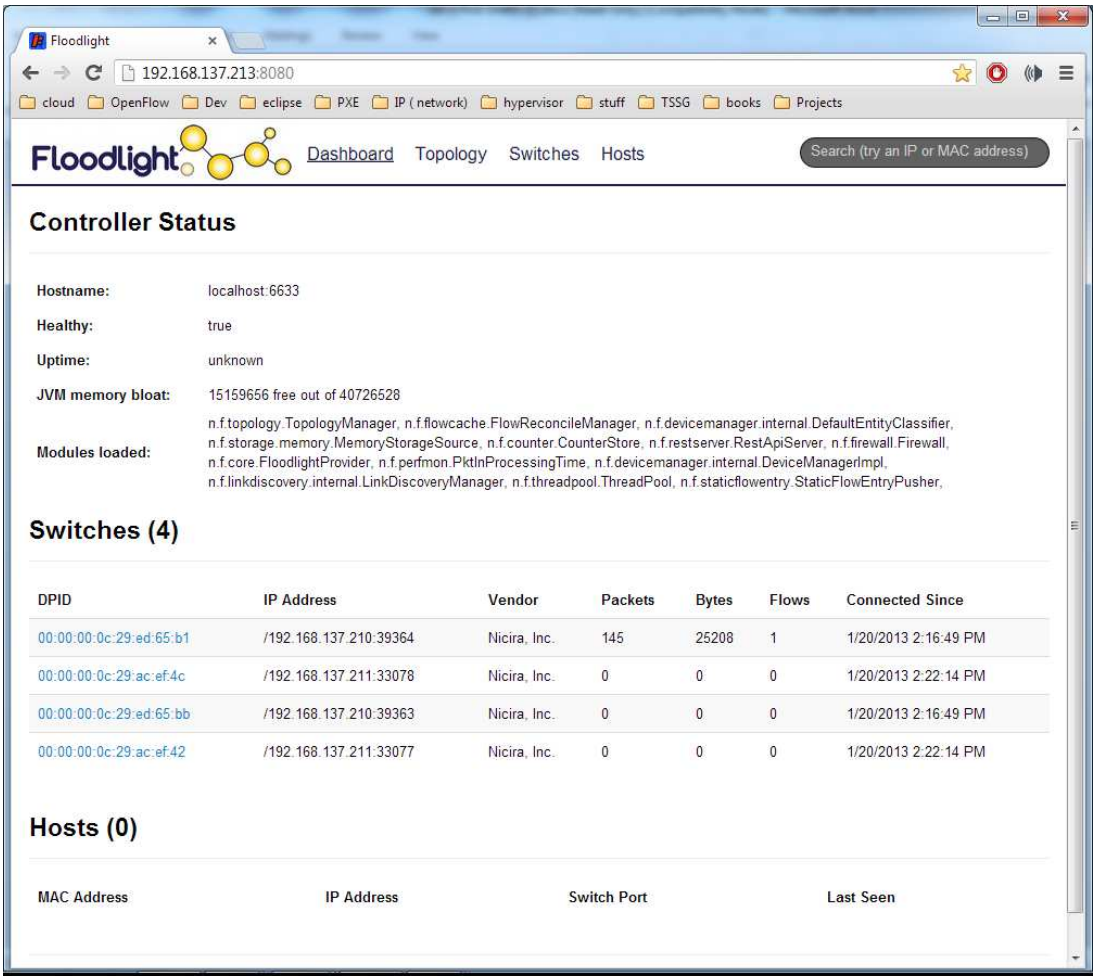


Figure 17: TSSG Floodlight OpenFlow Controller

The deployment of the GRE (Generic Routing Encapsulation) protocol as a tunnelling interconnect between the two sites (TSSG and UoP) allows the two partners to connect their testbeds Ethernet broadcast domains seamlessly. This extends the researchers and experimenters access to TSSG and UoP software defined network facilities. The GRE implementation is based on the OpenVSwitch solution and can be easily ramped up to incorporate other broadcast domains, further enhancing the user experience. As part of the WONDER project from the 2nd open call we are also looking into the possibility of interconnection at the IMS Network-SBC (Session Border Control) interface.

6 Task 1.5 "Enlarging the facility Consortium"

We refer the reader to Deliverable D1.5 [4], which as a matter of fact is entirely dedicated to providing a status on this particular matter, and that will be issued at the end of the Project by Month 30. This section is mentioned here only for compliance with the Description of Work [3].

7 Conclusion

WP1 is deemed essentially on track. Our achievements during this second year have allowed us to bring substantial improvements to all four of our technological focuses, namely again SFA, teagle, FRCP and MySlice.

Although the former two have reached a rather mature status in terms of design and development, the latter two are still under active development as of this writing. We intend to focus primarily on these two bricks over the third and last year of the project, with the following goals in mind :

- Promote FRCP as a big player in the Experimental Plane arena; again this paradigm currently already has been backed by big players like, of course, OMF, but also PlanetLab, NEPI, and other user tools, and we want to make this even more appealing;
- Improve MySlice even further; if measured simply in terms of functionalities, the progress of MySlice over this past year is modest. This is due to the major redesign tasks undertaken both on the backend and frontends. Now that these redesigns have been carried out we are confident to be able to attract more developers to build on top of this framework, and so to be able to come up next year with a feature rich user portal that will allow experimenters to seamlessly manage federation-wide experiments, especially if we manage to have MySlice take advantage of FRCP.

References

[1] Jordan Augé, Thierry Parmentelat, Nicolas Turro, Sandrine Avakian, Loic Baron, Mohamed Amine Larabi, Mohammed Yasin Rahman, , Timur Friedman, and Serge Fdida. Tools to foster a global federation of testbeds. *Computer Networks, Special issue on Future Internet Testbeds*, 2013.

[2] Loc Baron, Jordan Augé, Timur Friedman, and Serge Fdida. Towards an integrated portal for networking testbed federation, an open platform approach. *FIRE Engineering workshop*, Nov 6-7 2012.

[3] The OpenLab Consortium. Description of work, annex to contract, June 2011.

[4] Openlab d1.5: Second revised facility consortium agreement.

[5] Federated e-infrastructure dedicated to european researchers innovating in computing network architectures. http://www.fp7-federica.eu/.

[6] Architectural foundation for federated experimental facilities (omf6 design). http://mytestbed.net/projects/omf6/wiki/Architectural_Foundation.

[7] The global environment for network innovations. http://www.geni.net/.

[8] Alex Glikson. FI-WARE: Core Platform for Future Internet Applications. In *Proceedings of the 4th Annual International Conference on Systems and Storage*, pages ??–??, Haifa, 2011.

[9] Denis Havlik, Sven Schade, Zoheir a Sabeur, Paolo Mazzetti, Kym Watson, Arne J Berre, and Jose Lorenzo Mon. From Sensor to Observation Web with environmental enablers in the Future Internet. *Sensors*, 11(4):3874–907, January 2011.

[10] Alastair C Hume, Yahya Al-Hazmi, Bartosz Belter, Konrad Campowsky, Luis M Carril, Gino Carrozzo, Vegard Engen, David García-Pérez, Jordi Jofre Ponsatí, Roland Kbert, Yongzheng Liang, Cyril Rohr, and Gregory Seghbroeck. BonFIRE: A Multi-cloud Test Facility for Internet of Services Experimentation. In Thanasis Korakis, Michael Zink, and Maximilian Ott, editors, *Testbeds and Research Infrastructure. Development of Networks and Communities*, volume 44 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pages 81–96. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[11] Timur Friedman Jordan Augé. The open slice-based facility architecture (open sfa). http://opensfa.info/doc/opensfa.pdf, 2012.

[12] Nitos wireless testbed. http://nitlab.inf.uth.gr/NITlab/index.php/testbed.

[13] L Peterson, S Sevinc, J Lepreau, and R Ricci. Slice-based facility architecture. *Draft version*, 2009.

[14] Protogeni registry api v1.
http://www.protogeni.net/ProtoGeni/wiki/ClearingHouseAPI1.

[15] Very large scale open wireless sensor network testbed. http://www.senslab.info/.

[16] Sfa: Geni aggregate manager api version 2.
http://groups.geni.net/geni/wiki/GAPI_AM_API_V2.

[17] Sfa: Geni aggregate manager api version 3.
http://groups.geni.net/geni/wiki/GAPI_AM_API_V3.

[18] Christos Tranoris and Spyros Denazis. Federation Computing: A pragmatic approach for the Future Internet. In *2010 International Conference on Network and Service Management*, pages 190–197. IEEE, October 2010.

[19] Christos Tranoris and Spyros Denazis. FSToolkit: Adopting Software Engineering Practices for Enabling Definitions of Federated Resource Infrastructures. In Federico Álvarez, Frances Cleary, Petros Daras, John Domingue, Alex Galis, Ana Garcia, Anastasius Gavras, Stamatis Karnourskos, Srdjan Krco, Man-Sze Li, Volkmar Lotz, Henning Müller, Elio Salvadori, Anne-Marie Sassen, Hans Schaffers, Burkhard Stiller, Georgios Tselentis, Petra Turkama, and Theodore Zahariadis, editors, *The Future Internet*, volume 7281 of *Lecture Notes in Computer Science*, pages 201–212. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[20] G Tselentis. *Towards the Future Internet: A European Research Perspective*. IOS Press, 2009.

[21] University of thessaly. http://www2.uth.gr/main/index/index_en.html.

[22] Sebastian Wahle. *A Generic Framework for Heterogeneous Resource Federation*. Phd thesis, Technische Universität Berlin, 2011.

[23] Sebastian Wahle, Thomas Magedanz, and Konrad Campowsky. Interoperability in heterogeneous resource federations. In *Testbeds and Research Infrastructures. Development of Networks and Communities*, pages 35—-50. Springer, 2011.

[24] Sebastian Wahle, C. Tranoris, S. Denazis, A. Gavras, K. Koutsopoulos, T. Magedanz, and S.; Tompros. Emerging Testing Trends and the Panlab Enabling Infrastructure. *IEEE Communications Magazine*, 49(March):167–175, 2011.

[25] Sebastian Wahle, Christos Tranoris, Shane Fox, and Thomas Magedanz. Resource Description in Large Scale Heterogeneous Resource Federations. In *Testbeds and Research Infrastructure. Development of Networks and Communities*, pages 100—-115. Springer, 2012.