Wireless Software and Hardware platforms for Flexible and Unified radio and network controL

Project Deliverable D5.1

Design of experimentation tools

Contractual date of delivery:	30-06-2015
Actual date of delivery:	30-06-2015
Beneficiaries:	IMINDS, TCD, TUB, UFRJ, RUTGERS, NCENTRIC
Lead beneficiary:	IMINDS
Authors:	Pieter Becue (IMINDS), Vincent Sercu (IMINDS), Bart Jooris (IMINDS), Michael Mehari (IMINDS), Peter Ruckebusch (IMINDS), Nicholas Kaminski (TCD), Mikolaj Chwalisz (TUB), Ivan Seskar (RUTGERS), Robin Leblon (NCENTRIC), Jose F. de Rezende (UFRJ)
Reviewers:	Ingrid Moerman (IMINDS)
Work package:	WP5 – FED4FIRE compliance
Estimated person months:	3.5
Nature:	R
Dissemination level:	PU
Version:	3.5

Abstract:

This deliverable starts with an overview of all WiSHFUL testbeds and describes the status of their level of Fed4FIRE (F4F) compliance. Orbit and w-iLab.t are fully F4F compliant, while TWIST, IRIS and the FIBRE island at UFRJ are implementing their version of the federation AM API to be included in the F4F federation. Three F4F federation tools are described and will be extended during the course of the WiSHFUL project. SFA (jFed) will be used to discover, reserve and provision resources. FRCP (OMF6) will provide advanced functionality for experiment control. Finally, OMSP (OML) facilitates the collection of measurement and monitoring data. The deliverable is concluded by presenting tools for advanced wireless monitoring, a framework for the automation of experiments, a tool for the automation of complex software deployment and a short description of the integration of WiSHFUL UPI's in the testbeds.

Keywords:

Fed4FIRE architecture, testbed infrastructure, Fed4FIRE tools, WiSHFUL UPI tools

Executive Summary

This month 6 deliverable reports on the status of Fed4FIRE compliance of all WiSHFUL infrastructures. Some of the WiSHFUL testbeds have reached full F4F compliance (i.e. Orbit and w-iLab.t). The TWIST and IRIS testbeds are working towards an implementation of the federation AM API by using the Geni Control Framework. The FIBRE island at UFRJ uses the SFAwrap tool to enable an SFA compatible API. Of the available F4F federation tools, three tools are chosen to support experimenters on WiSHFUL testbeds: jFed, OMF6 and OML. jFed implements the SFA protocol and enables discovery, reservation and provisioning of testbed resources. OMF6 is an FRCP compatible tool that allows advanced experiment control by a uniform description of testbed resources and applications. The OML library provides an easy way for experimenters and facility providers to collect and store experiment measurement and monitoring data. Finally, some new tools are described to better assist users of the WiSHFUL testbeds and software in all stages of the experiment life cycle. Some advanced tools for wireless spectrum monitoring are described, followed by a tool for the automation of experiments. A framework for the automation of software deployment (Ansible) allows users to set up complex software suites on testbed resources in a uniform way. The deliverable is concluded by describing the integration of the WiSHFUL UPIs in a testbed environment.

List of Acronyms and Abbreviations

AJAX	Asynchronous Javascript And XML
AM	Aggregate Manager
AMQP	Advanced Message Queuing Protocol
AP	(Wireless) Access Point
ASIP	Application-specific instruction set processor
CH	Clearinghouse
COR	Channel Occupancy Ratio
EC	Experiment Controller
DHCP	Dynamic Host Configuration Protocol
DIFFS	DIgital Front-end For Sensing
DNS	Domain Name System
FLS	First Level Support
FRCP	Federated Resource Control Protocol
F4F	Fed4FIRE
GCF	GENI Control Framework
GENI	Global Environment for Network Innovations
GUI	Graphical User Interface
HRN	Human Resource Name
HTML5	HyperText Markup Language version 5
ISM	Industrial, scientific and medical radio bands
JSON	JavaScript Object Notation
LDAP	Lightweight Directory Access Protocol
LDIF	LDAP Data Interchange Format
LTE	Long-Term Evolution (known as 4G)
LTS	Long Term Support
ML	Measurement Library
MP	Measurement Point
MS	Measurement Stream
NDL	Network Description Language
OEDL	OMF Experiment Description language
OF	OpenFlow
OMF	cOntrol and Management Framework
OML	(Orbit) Measurement Library
OMSP	OML Measurement Stream Protocol

PDP	Policy Decision Point
PHP	PHP: Hypertext Processor (used to be Personal Home Page)
PKI	Public Key Infrastructure
PoE	Power over Ethernet
RC	Resource Controller
RDF	Resource Description Framework
REST	Representational State Transfer
RF	Radio Frequency
Rspec	Resource Specification
RSSI	Received Signal Strength Indication
SE	Sensing Engine
SCALDIO	SCAlable raDIO
SDR	Software Defined Radio
SFA	Slice-based Federation Architecture
SFI	Python command line client for SFA
SPARQL	SPARQL Protocol and RDF Query Language
SQL	Structured Query Language
SUMO	SUrrogate Modeling
TCP	Transmission Control Protocol
UN-II	Unlicensed National Information Infrastructure
URN	Uniform Resource Name
USRP	Universal Software Radio Peripheral
VLAN	Virtual LAN
VPN	Virtual Private Network
XML	Extensible Mark-up Language
XMPP	Extensible Messaging and Presence Protocol

Table of contents

1 Introduction

This deliverable reports on the work planned in WP5 “FED4FIRE compliance” and prepares the implementation work for the first experimentation toolset which is due on month 12.

The document first describes the ongoing efforts to get all WiSHFUL infrastructures compatible with the Fed4FIRE standards. A crucial part in the Fed4FIRE architecture is the use of SFA (Slice-based Federation Architecture). Therefore a short explanation of SFA is included in this deliverable, explaining concepts like slices, slivers, aggregate manager and clearinghouse. The Orbit testbed at RUTGERS and the w-iLab.t testbed at IMINDS are already compliant to Fed4FIRE. For the TWIST testbed at TUB and the IRIS testbed at TCD, it is described how they implement their version of the federation aggregate manager by using the GENI Control Framework package. The FIBRE island at UFRJ makes use of a different implementation to support an SFA interface on top of their testbed, namely the SFAwrap tool.

Out of the currently available tools offered by the Fed4FIRE project, a selection is made in order to support all WiSHFUL infrastructures and provide experimenters with a uniform way to access different types of resources. The experimenter tool that is chosen to enable experimenters to access the SFA interface of the testbeds is called jFed. jFed is a client-side experimenter tool that allows experimenters to discover, reserve and provision resources in a wide variety of testbeds. To support the FRCP standard for experiment control, the OMF6 tools are proposed in this deliverable. For the collection of experiment measurement data and the monitoring of both infrastructure and resources, the OMSP protocol and corresponding OML library are described. In the first stage of the WiSHFUL project, the focus will be on extending the jFed tool to allow experimenters to use resources of different testbeds in a uniform way. In later stages of the project, the OMF6 and OML will be extended to support more advanced experiment control and uniform measurement collection.

The final chapter of this deliverable is dedicated to new tools that will be offered by month 12 in the form of a first experimentation toolset. Because knowledge about the wireless spectrum is very important, the first category of tools is dedicated to advance wireless monitoring. Next, a tool for the automation of experiments is described which can be used by experimenters to explore multi-dimensional parameter spaces to obtain the optimal operating point(s) for a set of certain configuration parameters. If experimenters have complex software suites to be installed on testbed resources, the WiSHFUL project also offers an advanced tool for the automation of software deployment, called Ansible. To conclude the deliverable, a guideline is given for the integration of WiSHFUL UPIs into the testbeds.

2 Fed4FIRE compliance of WiSHFUL testbeds

This chapter describes how WiSHFUL testbeds are modified to comply with the Fed4FIRE standards. In the first section, the F4F architecture is briefly repeated (see D2.1) and a tool is presented that can facilitate the federation process of existing (or new) testbeds. The next sections describe how the federation process is being implemented for the existing WiSHFUL testbeds: TWIST (TUB), Router testbed (TUB), IRIS (TCD) and the FIBRE island at UFRJ. The chapter is concluded by shortly describing the Orbit testbed (RUTGERS) and the w-iLab.t testbed (IMINDS). These testbeds were already F4F compliant prior to the start of the WiSHFUL project. After successfully federating the testbeds listed above, all WiSHFUL testbeds will be F4F compliant.

2.1 Slice-based Federation Architecture (SFA)

The Fed4FIRE architecture is built around the concept of the Slice-based Federation Architecture (SFA[1], see D2.1). The SFA architecture covers the initial stages in the experiment lifecycle, such as resource discovery, requirements, reservation and provisioning (see left side of Figure 1). In order to comply with the F4F standards, every testbed needs to provide an SFA interface to support these stages. This is the case for both the advanced and light F4F federation model (see D2.1).

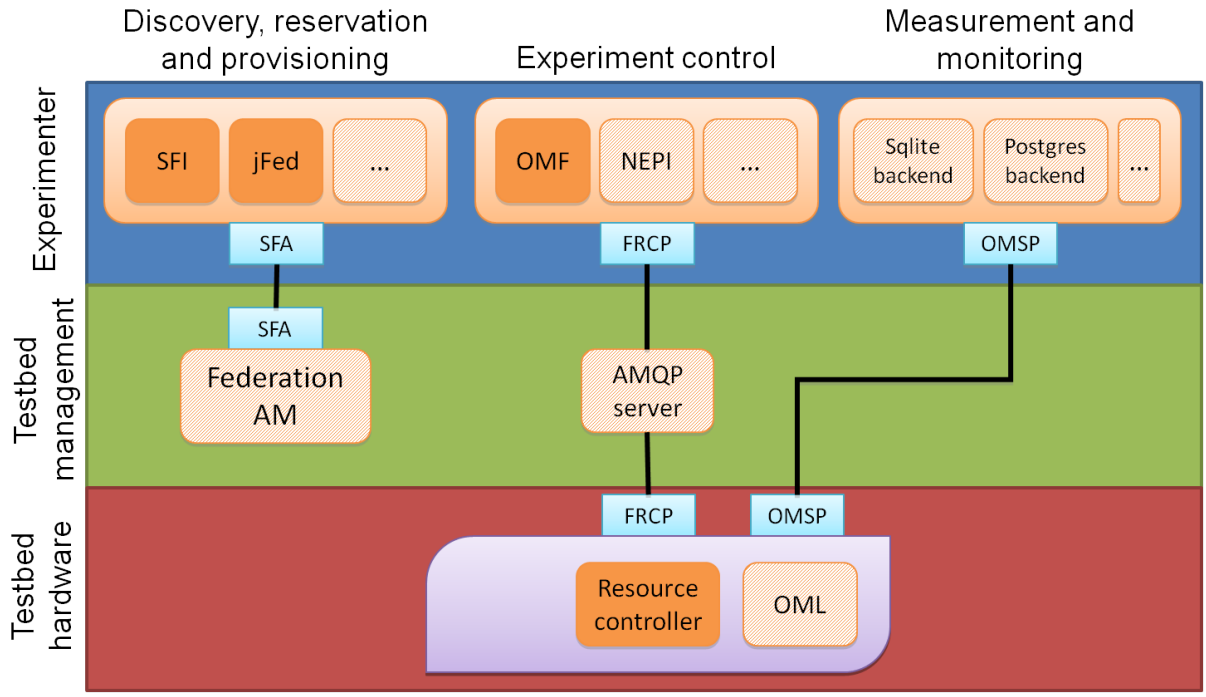


Figure 1 The Fed4FIRE federation tools.

Testbeds can enable an SFA interface by implementing two major components: a clearinghouse[2] and a Federation Aggregate Manager (AM) API[3]. In cooperation with GENI [4] (Global Environment for Network Innovations), the Fed4FIRE project offers several tools in order to make it easy for new testbeds to implement those components specifically for their testbed architecture. The tool chosen as a reference clearinghouse & AM implementation for the WiSHFUL testbeds is called the Geni Control Framework, now part of GENI-tools (see 2.1.3).

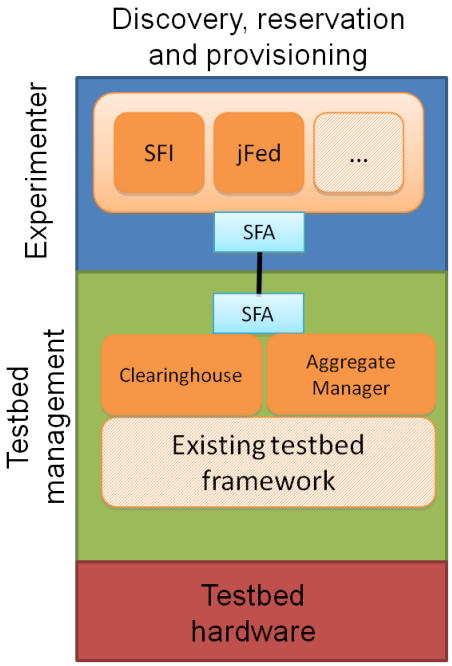


Figure 2 A Clearinghouse and AM example.

The clearinghouse (Figure 2) takes care of user, project and slice management, while the aggregate manager provides testbed resources to experimenters. The following two sections describe the functionality of the clearinghouse and the aggregate manager. The chapter is concluded by giving an overview of the Geni Control Framework.

F4F federation requirements such as documentation, policies and facility monitoring will not be described in this deliverable, as having an operational AM API is a crucial first step to joining the F4F federation. The federation protocols for experiment control (FRCP (see 3.2)) and measurement collection (OMSP (see 3.3)) are shortly described in the deliverable, but are not mandatory to join the F4F federation.

2.1.1 Clearinghouse

The functionality of the clearinghouse is split into the management of users, projects and slices.

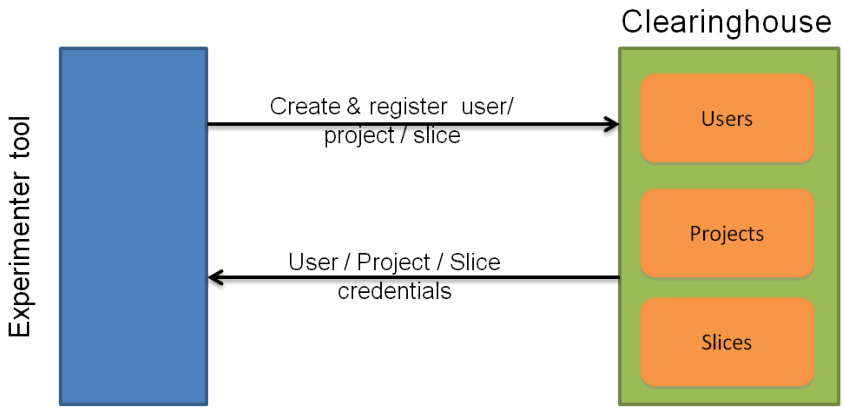


Figure 3 A Clearinghouse example.

The user management is handled by an Identity Provider and provides certificates and PKI (Public Key Infrastructure) keys to the experimenters.

The project authority is responsible for creating project and assigning roles to the different members of the project.

The slice authority provides experimenters with slice credentials. The slice concept is very important to understand the working of the SFA architecture: a slice is an abstraction for a collection of resources (WiFi/sensor nodes, servers, robots, etc.) capable of running experiments. A slice can be shared by one or more experimenters. An experiment can thus use resources of a certain slice. Experimenters can request slices at the clearinghouse and get a slice credential back.

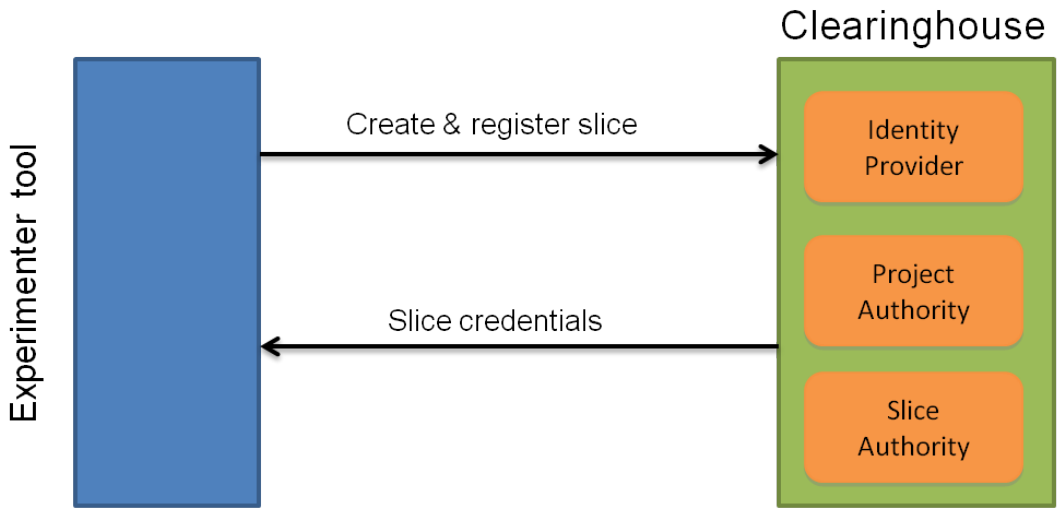


Figure 4 The get slice credential process.

Using this slice credential, the user can add resources to the slice. The functionality to add resources to a slice is part of the aggregate manager (see section 2.1.2). Therefore, a slice can contain resources from different testbeds at the same time. The selection of resources of a slice that are managed by the same aggregate manager is called a sliver.

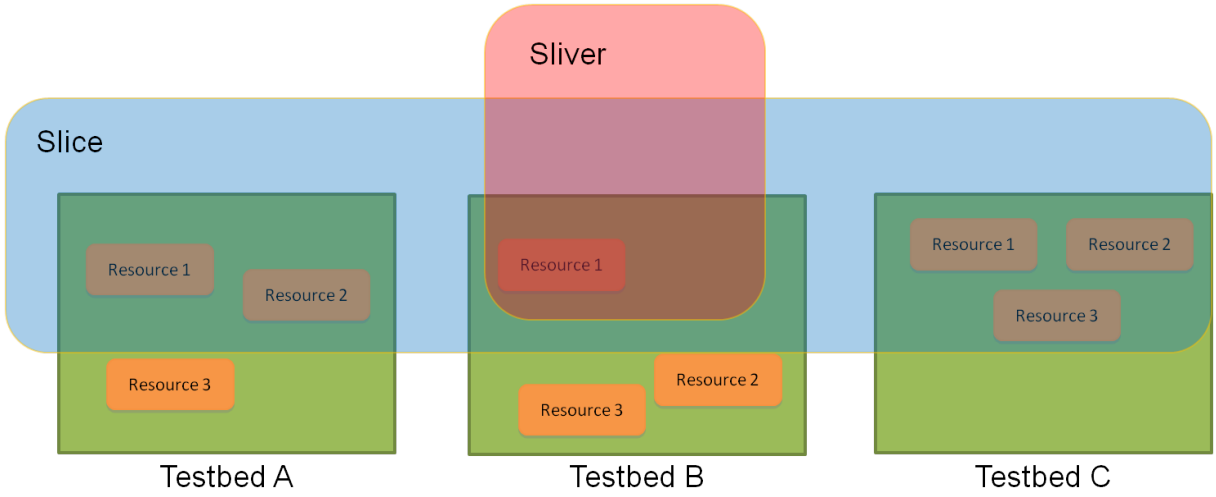


Figure 5 The Slice and sliver concept.

2.1.2 Aggregate Manager

The aggregate manager exposes the federation AM API. This API can be seen as an extra layer on top of an already existing testbed management framework. As with any API, the exposed functions are common across all testbed in the federation, but the implementation of each function will be specific for every testbed.

Experimenters can make use of SFA-enabled client tools (see section 3.1) to use the functions exposed by the AM API. Client tools communicate with AM APIs by using Rspec (Resource specification [5]) documents. Rspecs are XML-formatted documents that contain a detailed listing of resources (and their specifications) of a certain AM. There are three different kinds of Rspec documents:

- The advertisement Rspec is sent by the AM in response to a ListResources call and gives a detailed listing of all available resources managed by that AM.
- The request Rspec is passed as an argument to the CreateSliver call. It contains a subset of the resources selected by the user to run the experiment on.
- The manifest Rspec contains a detailed listing of the resources that have been reserved for the experimenter. This description contains all specifications of the resources that could be needed by the experimenter to conduct the experiment (e.g. network configuration, software versions, etc.).

The figure below shows a simplified version of the calls made by an experimenter tool to the aggregate manager.

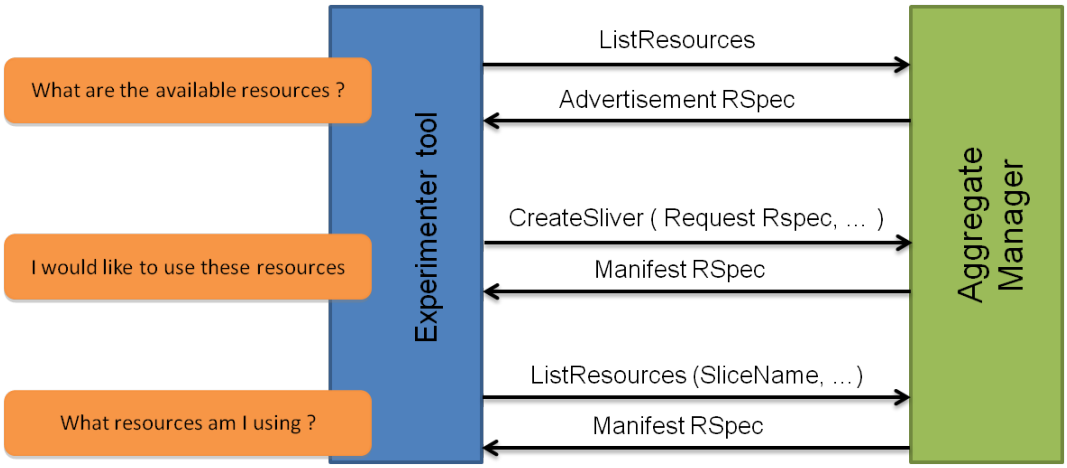


Figure 6 Available AM API calls.

The workflow to set up an experiment using the federation AM API is described below. The AM API functions that can be directly mapped onto a stage in the experiment lifecycle are also shown between brackets in the section below. Note that the CreateSliver command is split up into three separate calls in this description: Allocate, Provision and PerformOperationalAction.

The steps shown in Figure 6 and Figure 7 are explained in this section. The first step in the experiment lifecycle is to discover the available resources (ListResources), after which the experimenter will request a certain subset of those resources (Allocate). The next step is to provision the reserved resources (Provision) and start them up (PerformOperationalAction). After the resources are started, the status can be checked (Status) and the reservation can be extended (Renew). If the experimenter no longer has a need for the resources, they can be released (Delete). Different versions of the AM API are currently in use by several testbeds, therefore is it possible to request the AM API version from the tools (GetVersion) to ensure compatibility. Testbed owners can decide to stop (Shutdown) a slice (a collection of resources assigned to an experimenter) if the slice is in conflict with the testbed policies.

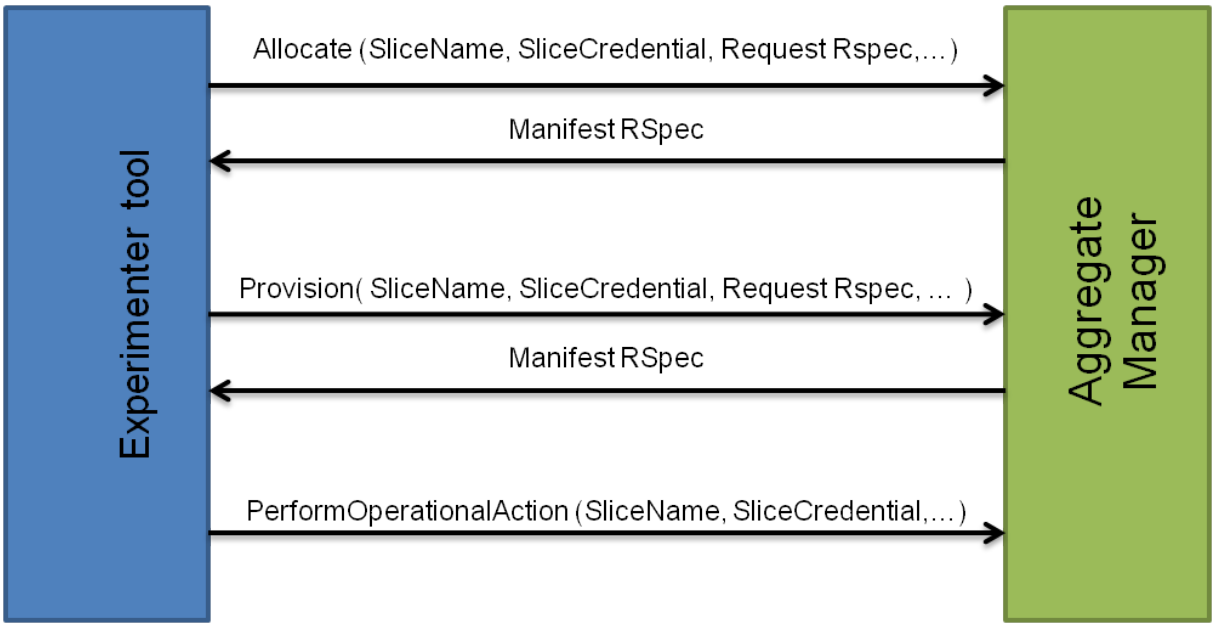


Figure 7 AM calls required for CreateSliver.

To conclude this section, the experiment workflow, including the AM API functions, are summarised below in the order in which they will be called for a typical experiment.

1. The experimenter gets a certificate and slice credential from the clearinghouse, thus renewing the slice if it had expired. The user and slice credentials are passed on as arguments for most of the AM functions below. They are omitted from the function definitions for the sake of simplicity.
2. The client tool executes a GetVersion call to learn which RSpec formats are supported by this aggregate manager.
3. The ListResources call invokes the sending of an Advertisement Rspec, describing all available resources at that aggregate.
4. The experimenter constructs a request Rspec, containing all the resources needed to conduct the experiment.
5. The Allocate(<request RSpec>) call invokes the following operations:
    a. The aggregate reserves the resources specified in the request Rspec
    b. The AM returns a manifest RSpec describing the reserved resources
6. Next, the Provision(<request RSpec>, <users struct>) call instantiates the resources and also returns a manifest Rspec describing the reserved resources and the configuration information that is specific for this instantiation. A part of the provisioning process could be to load an operating system onto a resource.
7. The Status(<slice URN or sliver URNs>) call checks if the resources are already provisioned.
8. Finally, the aggregate starts the resources in the slice by calling PerformOperationalAction(<slice URN>, ":start").
9. The Status(<slice URN>) call now checks if the resources have been started.
10. Optionally, the Renew(<slice URN or sliver URNs>, <new time>) call can extend the reservation of the resources.
11. The experimenter can now conduct the experiment and call Delete(<slice URN or sliver URNs>) when done.

2.1.3 GCF Software package

The Geni Control Framework (GCF [6]) software package implements a sample clearinghouse and aggregate manager. The GCF software package is part of GENI-tools on GitHub [7] and is being developed, extended and maintained by both the GENI and Fed4FIRE project.

The sample clearinghouse implementation takes care of all the specifics for dealing with certificates and PKI keys. The testbed administrator's tasks are limited to configuring some options and defining their policies (e.g. which AM's to trust).

The sample aggregate manager provides templates for all functions that have to be implemented in order to be compatible with other aggregate managers' APIs in the federation. The testbed specific implementation of the AM functions is the responsibility of the testbed administrators.

The GCF software package also includes some client tools, which can be used by facility providers to test their newly deployed clearinghouse and aggregate manager. However, in the first stages of the WiSHFUL project, only the jFed tool will be used (see 3.1).

To conclude, the GCF software package provides all the necessary tools for facility providers to implement their own federation AM API and should save them a considerable amount of time and effort to join the federation.

2.2 TWIST (TUB)

2.2.1 Testbed architecture

a. Sensor Network

The TKN Wireless Indoor Sensor Network Testbed (TWIST) is a multiplatform, hierarchical testbed architecture developed at the TKN. The self-configuration capability, the use of hardware with standardized interfaces and open source software make the TWIST architecture scalable, affordable, and easily replicable (Figure 9). The TWIST instance at the TKN office building is one of the largest remotely accessible testbeds with 204 sockets, currently populated with 102 eyesIFX and 102 Tmote Sky nodes (Figure 8). The nodes are deployed in a 3D grid spanning 3 floors of an office building at the Technische Universität Berlin (TUB) campus, resulting in more than 1500 m^2 of instrumented office space. In small rooms, two nodes of each platform are deployed, while the larger ones have four nodes. This setup results in a fairly regular grid deployment pattern with intra node distance of 3 m, as shown in Figure 10. Within the rooms the sensor nodes are attached to the ceiling.

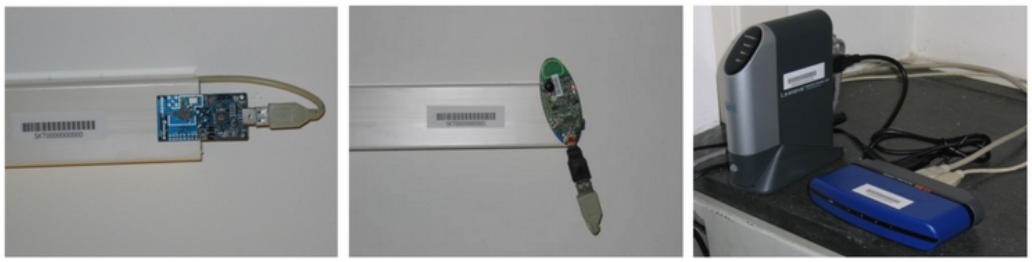


Figure 8 Tmote Sky (left), eyes IFXv2 (middle) and NLSU2 supernode / USB Hub (right).

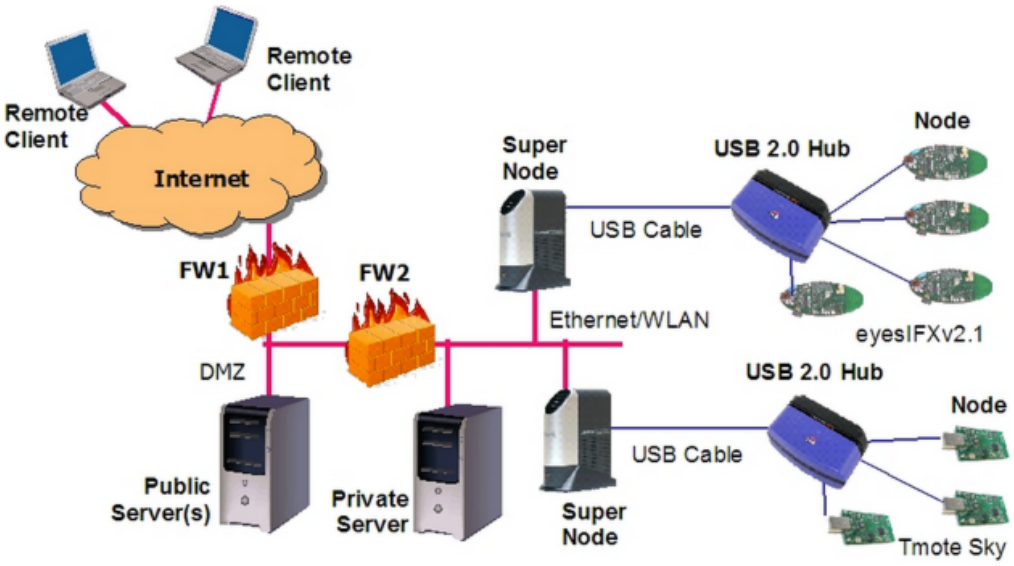


Figure 9 The hardware components of the TWIST testbed.

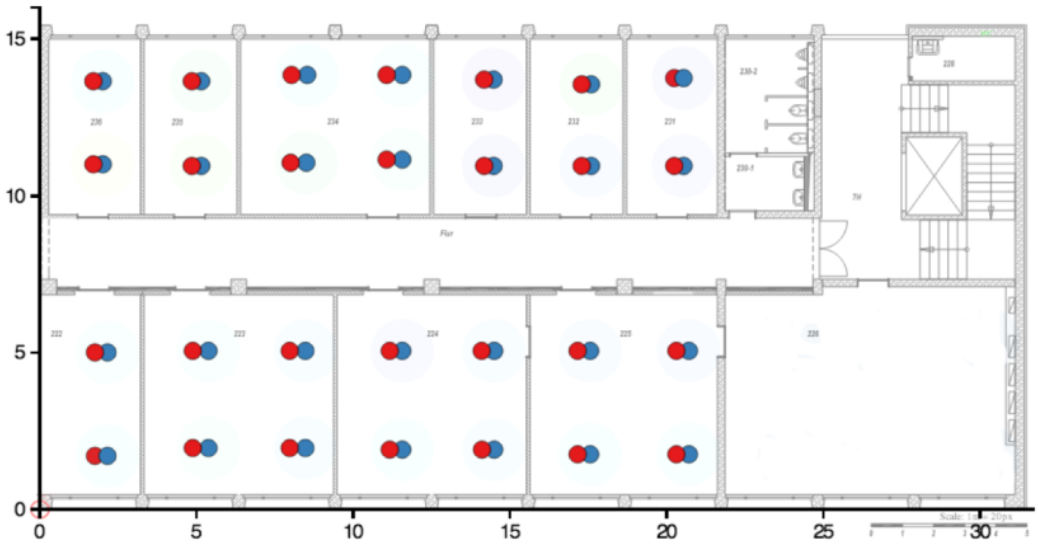


Figure 10 The locations of sensor nodes in the 2nd floor of TWIST testbed.

b. Turtlebot II Robotic Platform

Turtlebot II robotic platform comprises of a mobile base called Kobuki, a laptop, a router and a Microsoft Kinect 3D camera sensor (Figure 11). On the software side we are using Robot Operating System (ROS), an open source approach for robots. ROS comes with numerous drivers and libraries that cover everything from low-level communication with hardware as well as higher layer tasks, such as mapping, navigation and obstacle avoidance. Besides that, ROS is also a communication middleware that transports information between components in ROS. The dominating scheme is topic oriented asynchronous message exchange in the fashion of publish-subscribe, but it also has means for synchronous communication. It is easily extendible through either publishing or subscribing to existing topics or through creating new ones. By doing so, ROS can also be used to transport arbitrary data. This allows using ROS for controlling robots and extends the system by adding components on top of that.

We have set up an autonomous testbed environment in which we use the Turtlebot to position the Solution Under Test (SUT) at different locations (Figure 12). To do that we leverage the navigational capabilities of ROS that also includes obstacle avoidance. ROS uses a map, given a-priori, and localizes itself by matching the depth information of the Kinect 3D camera with the outline of the known map. ROS provides a simple interface to request the robot to drive autonomously to a given coordinate, so called goal, and a path planer is calculating the best path towards it. We have embedded these calls to move the robot to the next location into a higher schedule. First we define a set of waypoints that have to be covered in the experiment, then the robot iterates autonomously over each one of them. The whole procedure is followed in an unstructured office environment with dynamic obstacles, like humans, opening / closing the doors, etc.

For communicating with the rest of the infrastructure the mobile platform is equipped with a WLAN access point that operates in client mode and connects to one of the six APs deployed on every floor in our building. We are controlling the robot's AP by a ROS component that is location aware and selects the most appropriate AP in the different parts of the floor.

Figure 11 The Turtlebot Robotic Platform.

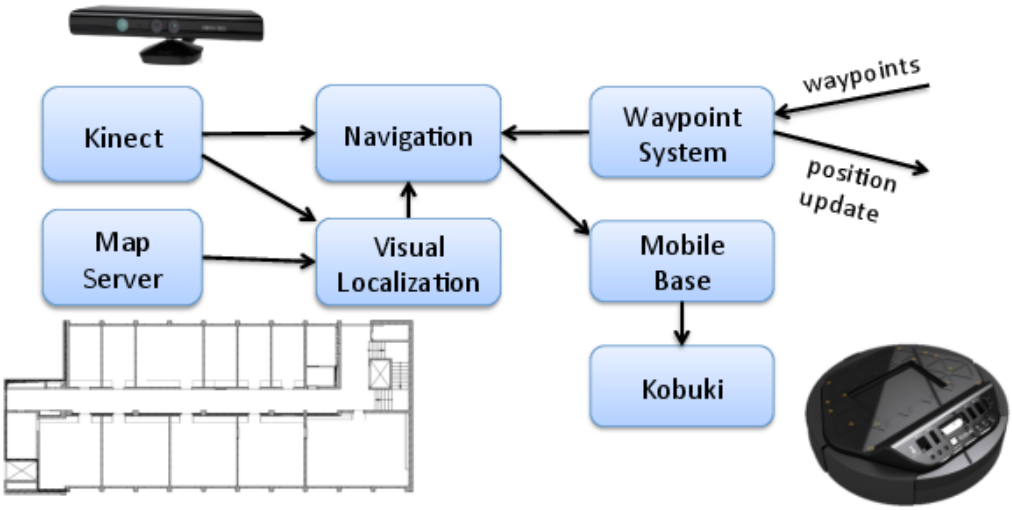


Figure 12 The Robotic Platform Design.

c. WLAN Access Points

TKN testbed is equipped with 18 dual band TP-link N750 APs (model TL-WDR4300) (Figure 13). They run under the control of the OpenWRT operating system that can be customized for each experiment. The positions of the WLAN APs in the 2nd floor of TKN testbed are depicted in Figure 14.

Figure 13 The TL-WDR4300 WLAN Access Point.

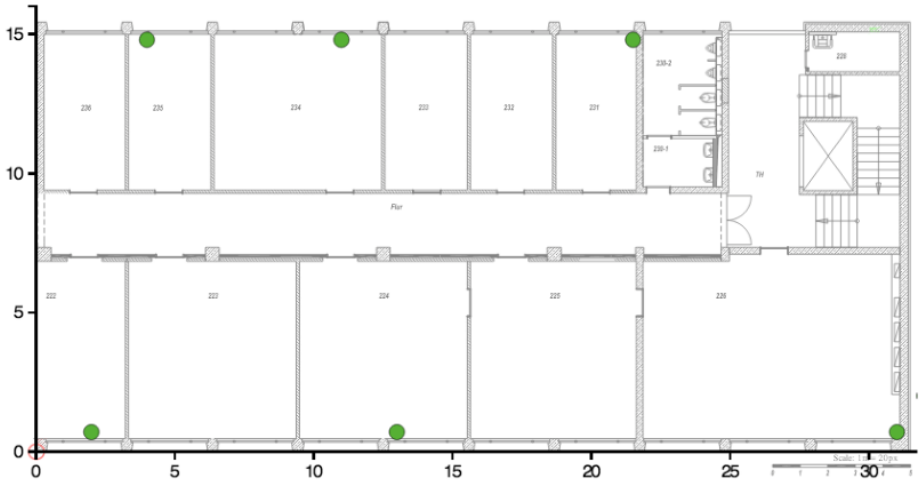


Figure 14 The locations of WLAN routers in the 2nd floor of TKN testbed.

d. WMP on Alix2D2 Embedded PCs

Wireless Mac Processor (WMP) is a customizable WLAN 802.11b/g MAC. It is running on ALIX2D2 embedded PCs equipped with Broadcom WL5011S 802.11b/g cards and shown in Figure 15. In our infrastructure three ALIX2D2 exist.

Figure 15 An Alix 2D2 embedded PC.

2.2.2 Federated testbed architecture

The WLAN routers, Robots, and Alix2D2 PCs, do not have a public API allowing external users to easily access them. However, to enable the access for external users, GCF will be used to implement the TWIST AM. It will be deployed in one of the TWIST public servers allowing for external federated users to provision resources in the testbed. This gateway server will interact with the internal infrastructure to configure the provided resources. For example, it will allow the users to deploy their own and custom OpenWRT image on the router as well as start and control the robot behaviour. It will also generate RSpecs, as needed to conform to the Fed4FIRE approach to user interaction. Finally it will provide SSH connections to the routers and API access to interact with the robot.

Currently TWIST Sensor Network runs under the control of its own open source control system allowing users to reserve time on the testbed on the per platform basis. It is a stable system working for many years but in the current state it is not compatible with the AM APIs. It is planned that in the first phase of the work this system will stay in place without any modifications. The TWIST AM will in the later stage provide access to the sensor network in terms of image deployment on the particular sensors and Serial Forwarder interface to communicate with them. This can be achieved by a direct mapping to the current TWIST access to the sensor nodes. Those are constrained devices, so SSH based access cannot be achieved.

2.3 IRIS (TCD)

2.3.1 Testbed architecture

The Implementing Radio In Software testbed (IRIS) is a reconfigurable radio testbed based on software-defined radio (SDR), hosted by Trinity College Dublin (TCD), through the telecommunications research centre (CTVR). This testbed supports experimentation with a flexible radio system running on a virtualized computational platform. The testbed is organised into 16 experimentation units, each of which consisting of three parts: a virtual computational platform, SDR software, and flexible radio frontend hardware. This organization encapsulates the elements required to use an SDR system to construct a broad range of radio systems. Each experimentation unit is designed to flexibly serve a range of needs: Linux (Ubuntu 14.04 LTS) provides a highly configurable computation platform, IRIS provides real-time radio reconfigurability, and a Universal Software Radio Peripheral (USRP [8]) offers a broad range of wireless interfaces. Radio hardware is housed on the ceiling of a dedicated indoor testing space to provide users with a clean operating environment. The management infrastructure allows users to deploy experimentation units to compose arbitrary radio systems and networks as desired.

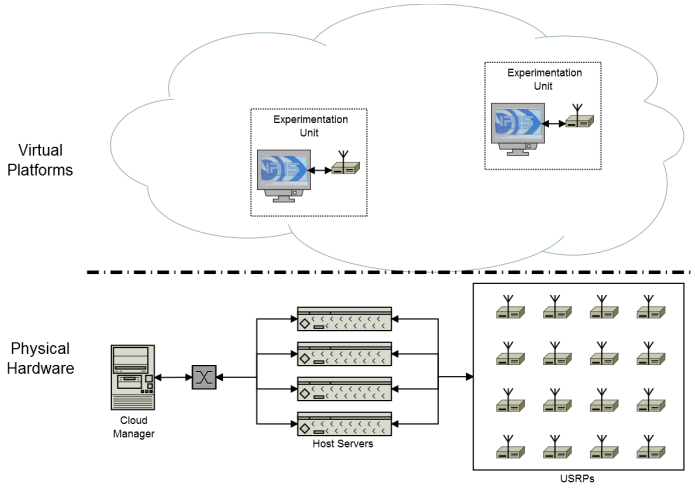


Figure 16 The TCD testbed System Architecture.

Figure 16 displays the architecture of the virtualized testbed. In this paradigm, underlying hardware is composed into experimentation units to server users as described above. An array of servers, referred to as host servers, provide the computational power to support running SDR software and supporting software with virtual machines. Each virtual machine is connected to a USRP mounted on the ceiling grid within the dedicated testing space. The cloud manager coordinates and controls virtual machines, handling the deployment of computational environments and their connection to USRPs.

2.3.2 Federated testbed architecture

Figure 17 displays the application of the GCF based AM in the TCD testbed. The AM implementation is deployed on a publically accessible gateway that allows federation users to provision resources as necessary. The GCF implementation is used as a frontend to the facility that handles requests from federated users in the manner specified by the Fed4FIRE project. These requests are translated into an internal TCD testbed protocol to interact with the cloud manager that configures the underlying resources to provide the requested platforms. The gateway server also generates RSpecs as needed to conform to the Fed4FIRE approach to user interaction. Finally, the gateway server also provides SSH connectivity to experimentation units once they are provisioned.

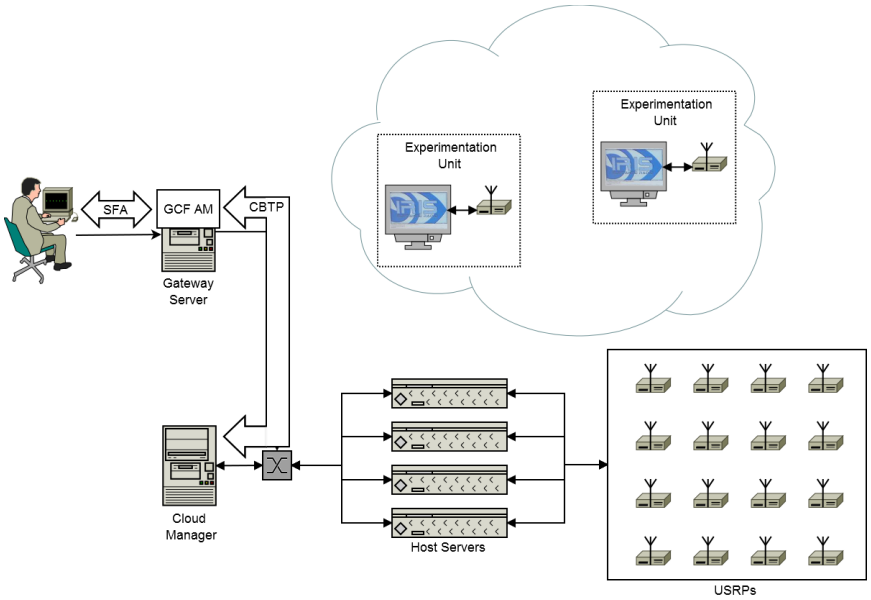


Figure 17 The TCD testbed user interaction.

The GCF based AM exposes resource provisioning in the standard Fed4FIRE manner. For example, the ListResource call returns an Advertisement Rspec, listing all of the currently available experimentation unit versions and associations (a VM connected to a USRP, where the version refers to the pre-loaded version of IRIS and the position refers to the position of IRIS on our grid). The Provision command then interacts with the Cloud Manager to start VMs loaded with the appropriate version of IRIS and connect them to the selected USRP. In this way the GCF AM exposes resource provisioning functionality through SFA.

2.4 FIBRE Island (UFRJ)

The FIBRE testbed island at UFRJ is being set up in the FP7 FIBRE-EU project. The UFRJ testbed forms one of the testbed Islands of FIBRE facility infrastructure at the Brazilian side. The hardware of the UFRJ testbed is shown in Figure 18.. The testbed consist of three NetFPGA devices, 8 wireless Icarus

nodes and an IBM server to host some virtual machines and take care of some testbed management functions such as LDAP. An OpenFlow switch acts as the FIBREnet border router.

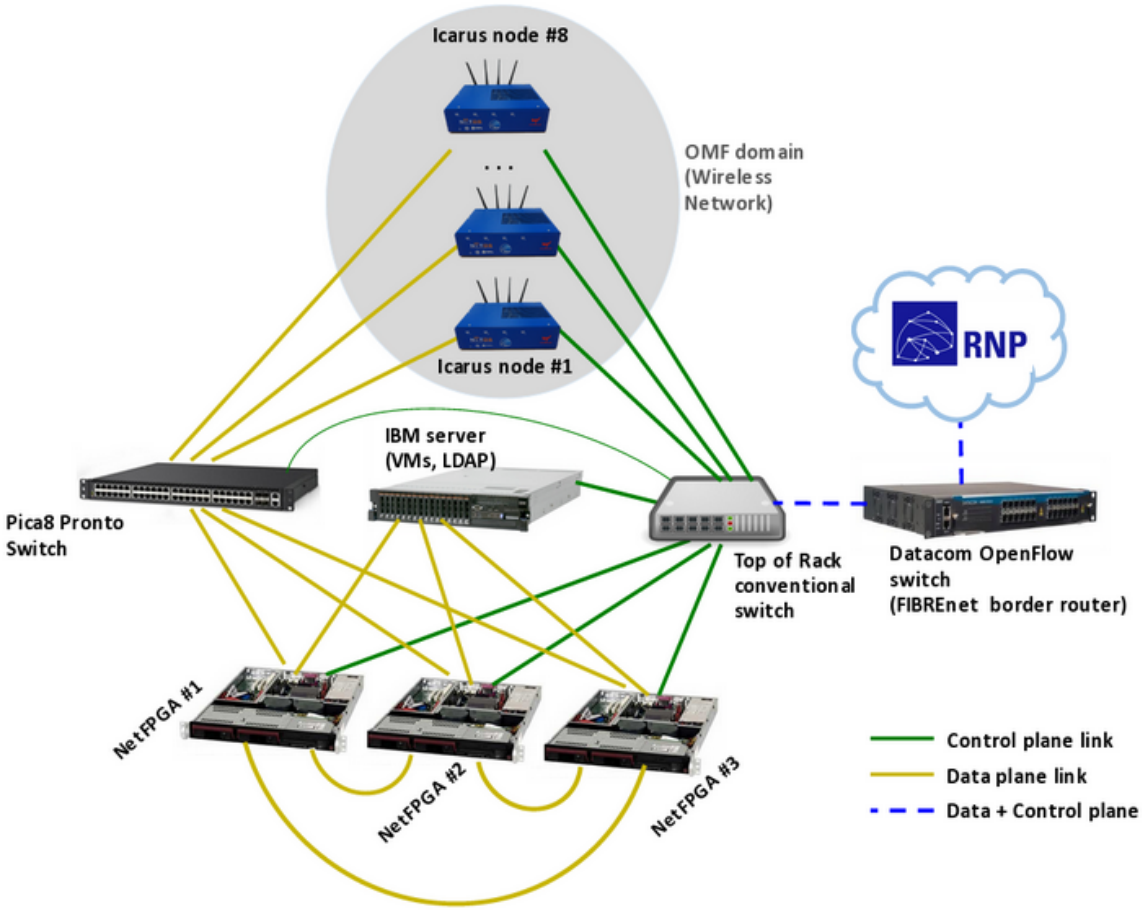


Figure 18 A FIBRE island with UFRJ hardware.

This testbed makes use of a different implementation of the SFA-layer, namely SFAwrap [9]. The FIBRE architecture is shown in **Error! Reference source not found.**. Users can create an account and control their experiments through the MySlice [10] portal. The MySlice portal translates the user input into SFA calls that are made to either the SFA registry or the SFA AM API's (AMv3) of the testbeds that are part of FIBRE. The SFA AM API's were developed based on the SFAwrap [9] implementation and are therefore F4F compliant.

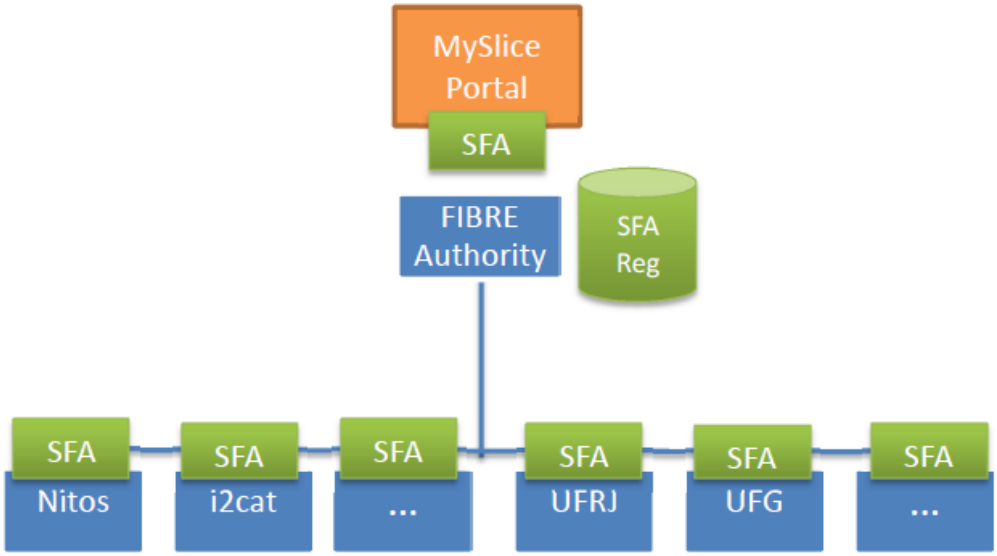
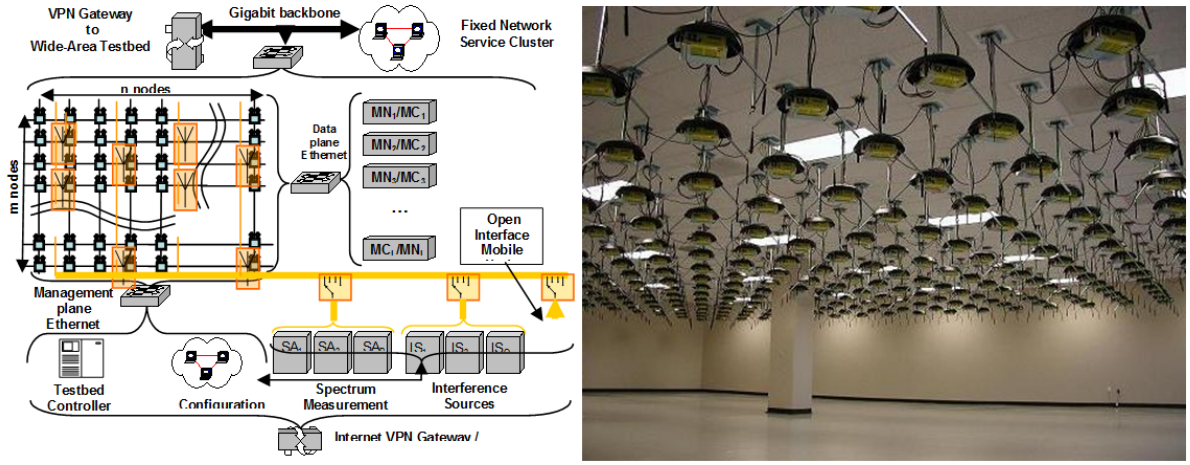


Figure 19 The FIBRE architecture.

2.5 Orbit (RUTGERS)

The 400-node ORBIT radio grid testbed at WINLAB, RUTGERS University is shown in Figure 20 below. The testbed provides 400 programmable radio nodes for at-scale and reproducible emulation of next-generation wireless network protocols and applications. The ORBIT radio grid can be accessed by experimenters via an Internet portal, which provides a variety of services to assist users with setting up a network topology, programming the radio nodes, executing the experimental code, and collecting measurements. The testbed also supports end-to-end wired and wireless experiments using a combination of ORBIT and OpenFlow switch/router nodes under the same experimental execution framework. In addition to fixed-function wireless devices (802.11 a/b/g/n/ac/ad, Bluetooth, ZigBee, WiMAX, LTE, etc.), the testbed has a number of SDR platforms including URSP 1/2/N210/B210/X310, WARP, CRKIT, Nutaq ZeptoSDR/PicoSDR and ZedBoards to support programmability at the radio PHY and MAC layers as needed to support emerging cognitive radio networking experiments. The radio grid is also supplemented by a number of sandboxes, outdoor and vehicular nodes (both WiFi and WiMAX) deployed on or around the RUTGERS campus, to be used for real-world validation of results or for application trials.



(a) ORBIT radio grid architecture (b) ORBIT radio grid at RU tech center building

Figure 20 The ORBIT Wireless testbed.

In addition to main ORBIT facility, 10 mini ORBIT testbeds (service machine with 3 nodes) with a set of 26 WiMAX/LTE base stations are being used to support wireless aspects of the GENI on selected campuses. OMF (version 5.4bis), as one of the control frameworks in GENI, is used as the main control/management framework in all of these deployments.

While strictly speaking, ORBIT OMF version does not support GENI standardized SFA directly, it is fully integrated with the GENI portal (and indirectly with other federated testbeds) through OMF GENI Portal AM which is used for the distribution of Authentication and Authorization (A&A) information from the portal to individual sites in the form of LDAP (LDIF) records (ORBIT site is the root site in the distribution tree for all GENI wireless OMF deployments). These LDAP records are then used for basic SSH authentication as well as for access to individual site WEB-based schedulers (these are "single users" sites and access to each site is controlled through OMF Scheduler AM service).

Features and capabilities of ORBIT (as well as of each of the GENI wireless testbeds) are exposed through the OMF Inventory AM service which allows discovery in two forms:

1. REST-like query with XML output: the aggregate/node/device capabilities are returned by the service as a structured XML response.
2. SPARQL query with XML output: a number of existing domain specific networking ontologies such as NOVI and NDL as well as a number of newly developed wireless and testbed ontologies (that were introduced through collaboration with the CREW project) are used as an input for the community driven "Testbed as a Service Ontology Repository" (TaaSOR). The resulting abstraction layer over heterogeneous testbeds is then used for testbed-independent queries.

2.6 W-iLab.t (IMINDS)

The w-iLab.t testbed [11] is deployed in Zwijnaarde, Belgium. It consists of 60 node locations. Every location hosts an embedded PC with two 802.11a/b/g/n Wi-Fi interfaces, an in-house designed 802.15.4 sensor node and a Bluetooth dongle. In addition to the 60 fixed wireless nodes, the w-iLab.t also hosts 20 mobile nodes. These mobile nodes can be configured and used in exactly the same way as the fixed nodes, but have the capacity to change location during the experiment. The w-iLab.t also supports the new 802.11ac. Currently, 10 fixed wireless nodes and all mobile nodes are equipped with an 802.11ac Wi-Fi card. Next to the embedded PCs, the w-iLab.t can also be used to conduct cognitive radio experiments by using USRP's (x8), WARP's (x3), ZedBoard (x8) or imec sensing engines

(x6). To support LTE experiments, 4 femto cells are installed and about 30 LTE client USB dongles (on both fixed and mobile nodes).

The w-iLab.t uses a combination of three management frameworks:

- Emulab [12]: This framework takes care of resource discovery, reservation and provisioning. It allows experimenters to design complex topologies and transparently takes care of all network configurations. The Emulab framework also provides a GENI AMv3 SFA interface (see 2.1). Experimenters can choose to use the testbed by either using the native Emulab interface, or the SFA interface in combination with fed4FIRE tools.
- OMF6: the OMF6 framework is used for doing experiment control (see 3.2.1).
- OML: the OML framework is used for the collection of experiment results, measurement data and for both infrastructure and experiment monitoring (see 3.3.1).

The architecture of the w-iLab.t testbed is depicted in the figure below.

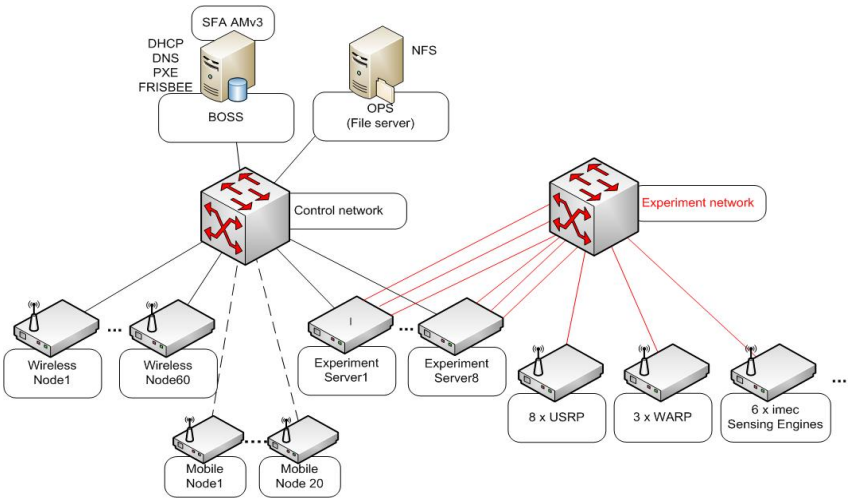


Figure 21 Emulab deployed at w-iLab.t.

The main server of the Emulab framework is called the BOSS server. This server is responsible for tasks such as DHCP, DNS and the loading of operating systems onto the testbed nodes. The BOSS server also provides an SFA AMv3 [13] interface to the outside world. This functionality comes pre-installed with the Emulab software package. The OPS server is the file server where users can store their data.

The deployment of the OMF framework in the w-iLab.t testbed is depicted in the figure below. Note that there are 2 extra servers needed: one AMQP server and one optional OMF experiment controller. The experiment controller doesn't have to be provided by the testbed, since users can install their own OMF experiment controller on one of the testbed nodes. Next to these servers, a ruby daemon, called the OMF Resource Controller has to be installed on every testbed node. For more information on the OMF framework, please see section 3.2.1).

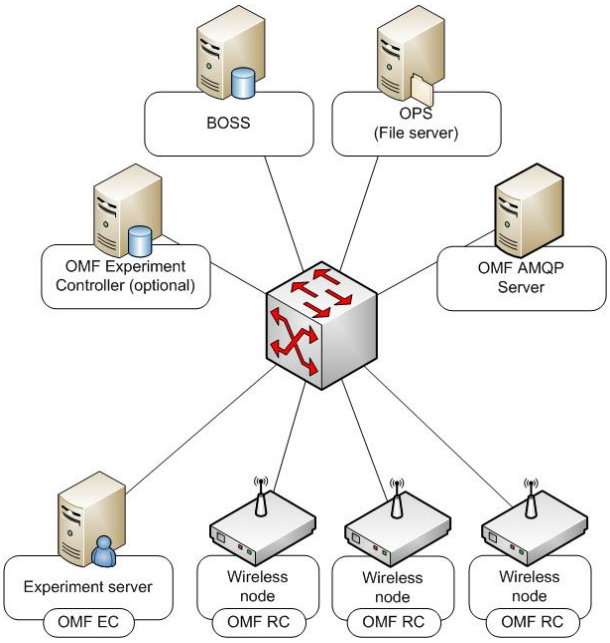


Figure 22 OMF deployed at w-iLab.t.

To support OML, the w-iLab.t testbed offers one publicly available OML server. All testbed nodes are pre-installed with the OML client library. Section 3.3.1 describes the OML framework in more detail.

3 Deployment and extension of existing F4F tools

This chapter describes the three federation protocol as supported by the Fed4FIRE project: SFA, FRCP and OMSP. In the next sections, every protocol is shortly described, followed by one example of a client tool that supports these protocols. In the first stage of the WiSHFUL project, focus will be on these client tools only. Based on feedback from open call experimenters or experiences from the project partners, other tools might be chosen or modifications will be made to the existing tools.

3.1 SFA

The SFA protocol takes care of resource discovery, requirements, reservation and provisioning. A detailed description of the protocol is given in section 2.1.

3.1.1 jFed

The jFed tool [14] is a client side SFA tool to enable experimenters to easily set up their experiments across different testbeds in the federation. The architecture of the tool is shown below.

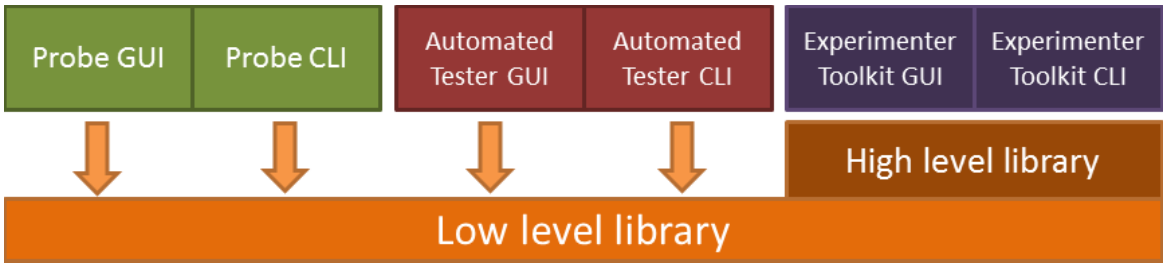


Figure 23 The jFed suite architecture.

The suite is built around the low level library, which implements the client side for all the supported APIs (e.g. GENI AMv3); and a high level library, which manages and keeps track of the lifecycle of an experiment. On top of these libraries various components were developed to allow thorough examination and testing of these APIs, as well as an user-friendly graphical Experimenter GUI to allow end-users to use the testbeds.

The most important components are:

- jFed Experimenter GUI allows end-users to provision and manage experiments.
- jFed Probe assists testbed developers in testing their API implementations
- jFed Automated tester performs extensive fully-automated tests of the testbed APIs, in which the complete workflow of an experiment is followed. This tool is used as part of the Fed4FIRE testbed monitor.

The remainder of this chapter briefly describes the jFed Experimenter GUI. For more information on the jFed tool, see Fed4FIRE D2.4 (http://www.fed4fire.eu/deliverables/).

jFed is written in Java. It uses the credentials and certificates issued by the clearinghouse (see 2.1.1) to authenticate and authorize users on testbeds. The experimenter can create new experiments by using the drag & drop functionality to select different types of resources. The current selection of node types contains Generic nodes (jFed will choose a node), Physical nodes (a complete physical server, no virtual machine) and three types of virtual machines (generic, XEN [15] or OpenVZ [16]). Also wireless nodes and channels can be reserved. A dedicated external network connection can be used to setup layer 2 connectivity between testbeds.

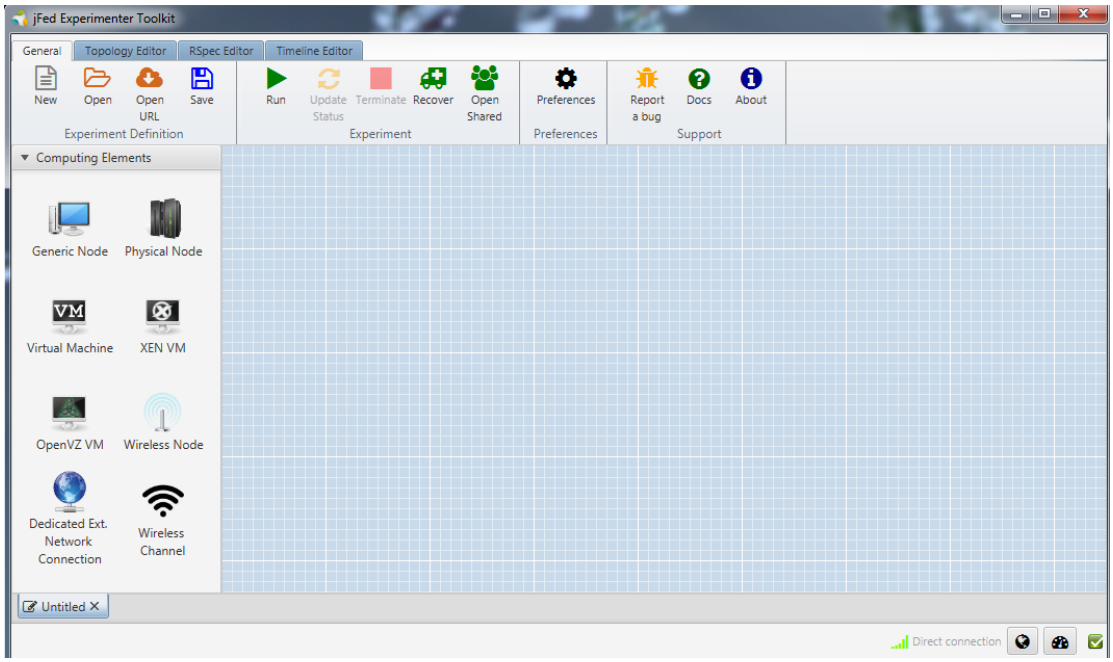


Figure 24 A new jFed experiment.

After dragging in some nodes, the user can select on which testbed he wants the experiment to run. Extra configuration can be done such as selecting a specific node, changing the default operating system and select a script to be run when the node boots. Network connections can be drawn between nodes, which will invoke the creation of a VLAN on the experiment switch of the testbed.

The example below shows an experiment on the w-iLab.t testbed, which uses two wireless nodes (named AP and client), one USRP and one server to process the results produced by the USRP. The server and the USRP are connected to each other by drawing a link between them.

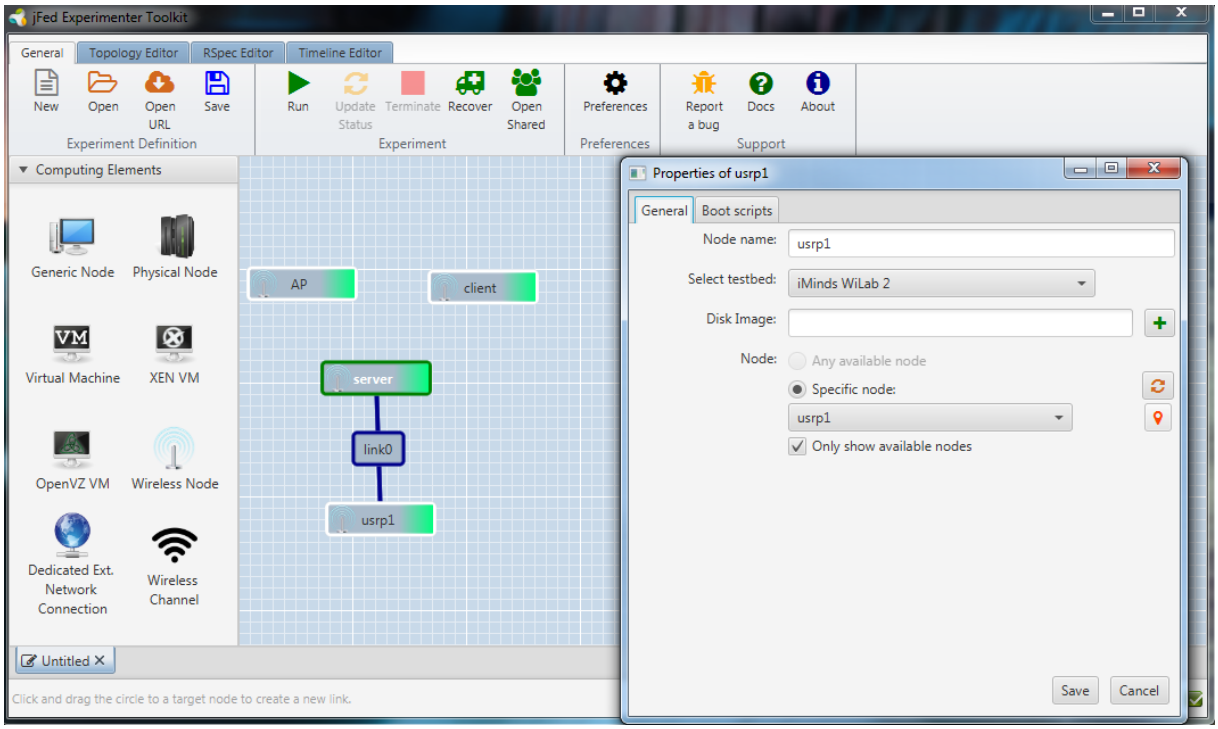


Figure 25 A jFed wireless experiment.

When starting the experiment, jFed sends a request Rspec to the correct testbed aggregate managers to start the allocation and provisioning of the resources. The request Rspec can be viewed on the Rspec editor tab. An example Rspec is shown here for the wireless experiment as shown above.

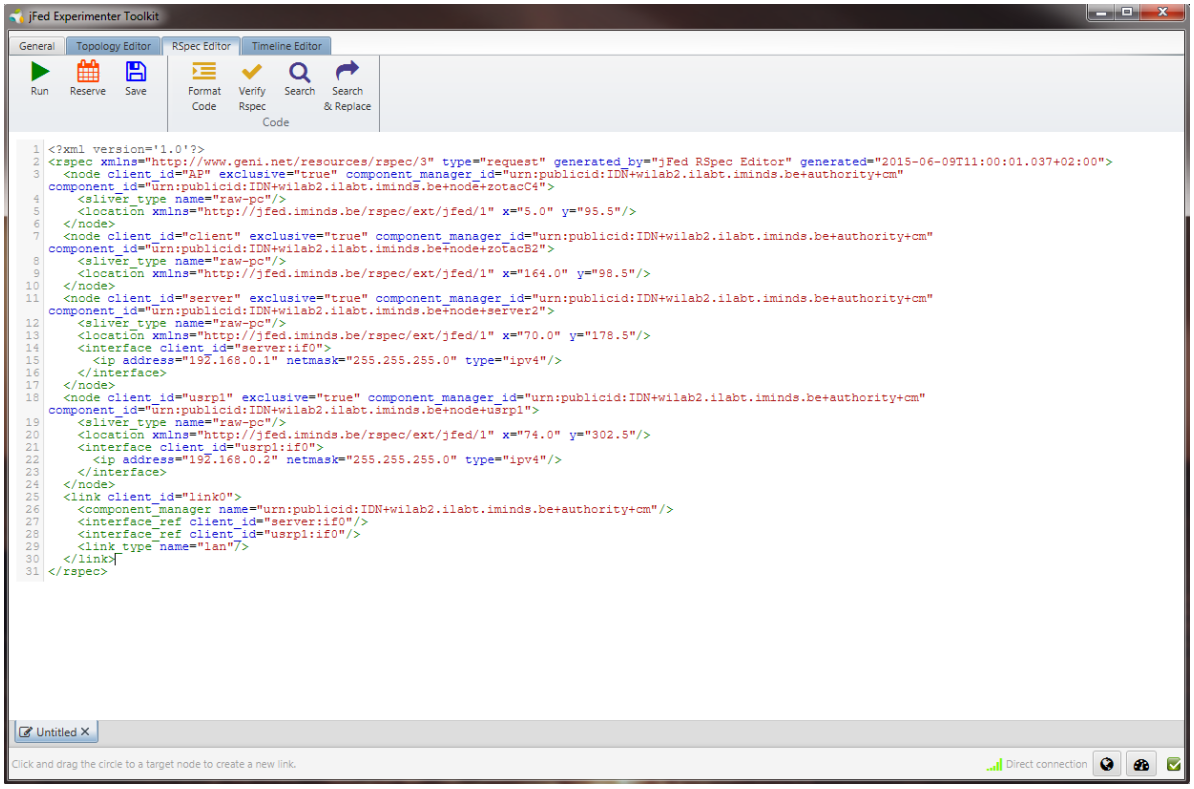


Figure 26 The jFed Rspec editor.

The jFed tool can also be used to get an understanding of the working of the SFA protocol. By clicking the icon on the bottom right, an overview is given of the calls made by jFed to the testbed AM's. The example below shows the calls made to get an overview of the available resources at the w-iLab.t testbed.

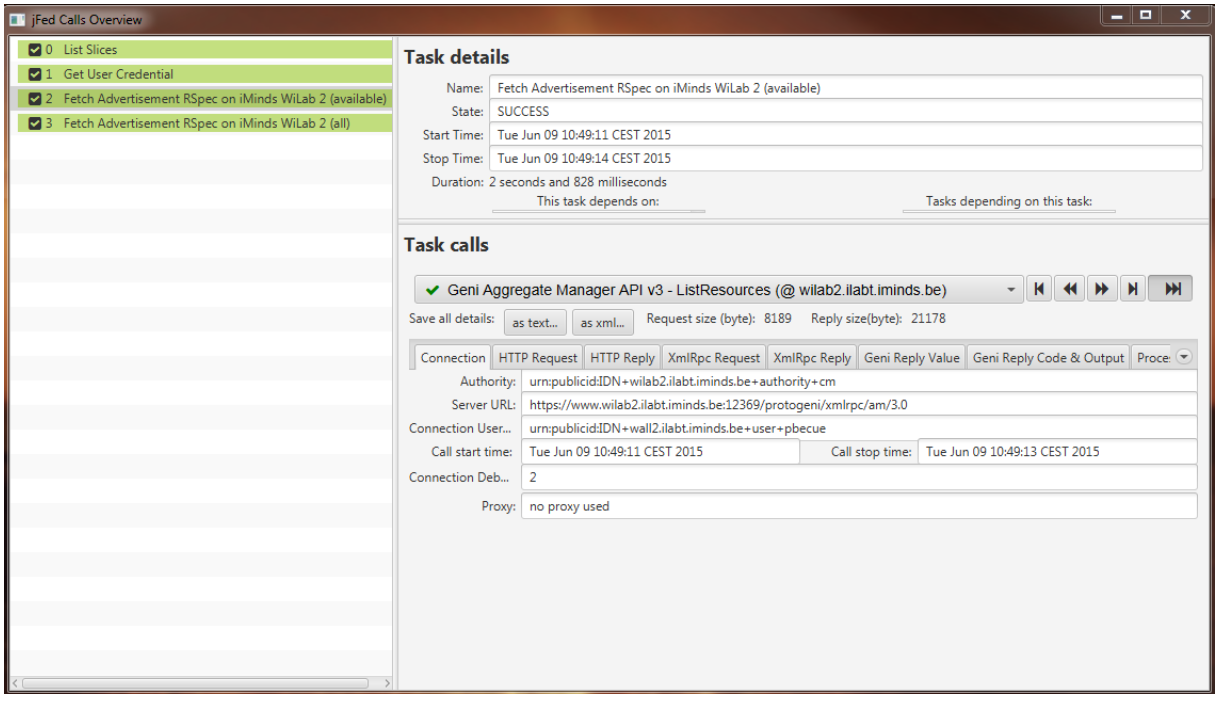


Figure 27 jFed SFA calls.

The jFed tool will be extended during the course of the project to support the testbeds in WiSFHUL. This will happen in close collaboration with the Fed4FIRE project. More specifically, jFed will be extended to parse Rspec documents specific for WiSHFUL testbeds and have the contact information for all the aggregate managers. The possibility to use jFed in offline modus to control the portable testbed (see D6.1) will also be investigated.

3.2 FRCP

The Federated Resource Control Protocol [17] is used to control and orchestrate distributed resources, such as testbed devices, sensor nodes or measurement software. The protocol can use messages in XML or JSON format. The messages are transported by using a Publish/Subscribe mechanism like XMPP [18] or AMQP [19].

The basic protocol consists of messages 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. The protocol consists of five messages: inform, configure, request, create, and release. An example of the use of these messages is shown in the figure below. A requester can create, configure and release resources. As response to these messages, the resources inform the requester with its status. The requester can also request the status of the resources with a separate message. These messages form the core of the FRCP protocol.

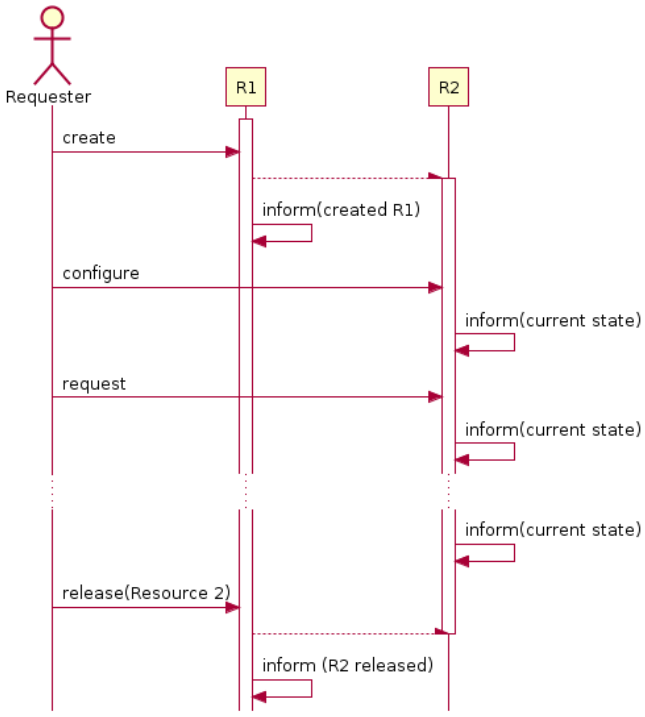


Figure 28 FRCP example.

The Fed4FIRE architecture regarding experiment control is shown below.

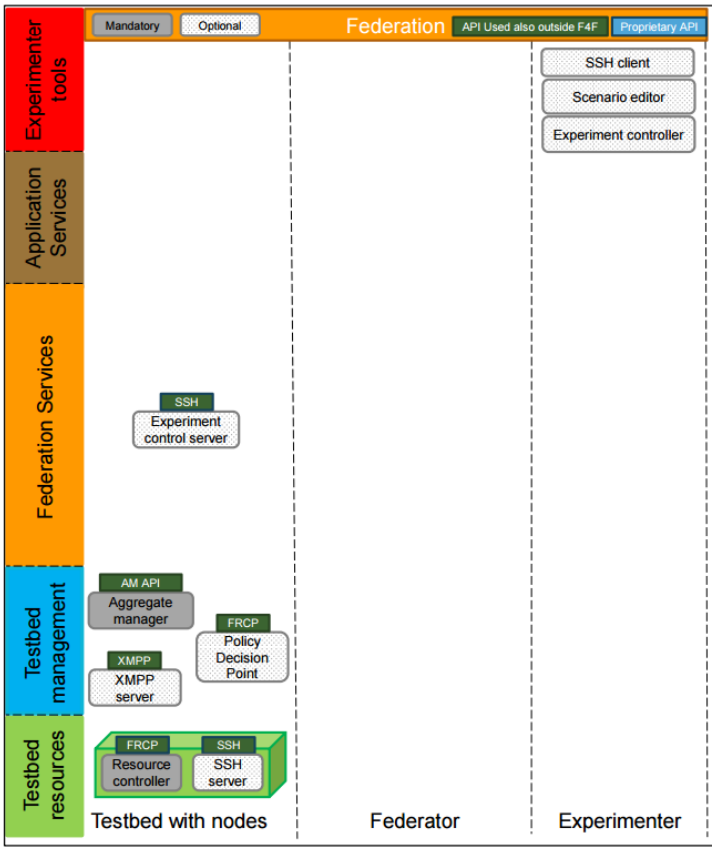


Figure 29 Fed4FIRE experiment control.

Testbed resources can either be controlled by SSH or by an FRCP enabled tool like OMF6 (see 3.2.1). The experimenter does not interact directly with the FRCP API on the testbed resources, but submits an experiment description to an experiment controller. This experiment controller translates the experiment into FRCP message, which are then sent over an XMPP or AMQP server towards the testbed resources. To authenticate and authorize experimenters to use certain resources, a Policy Decision Point (PDP) is introduced. The PDP queries which nodes belong to which slice and decides which experimenter has the rights to control these nodes.

A detailed description of the FRCP protocol is out of scope of this document. For more information, please see Fed4FIRE D2.4.

3.2.1 OMF6

OMF6 [20] is a control and management framework for testbeds. It implements the FRCP protocol.

In an OMF testbed, everything is a resource. The entity that controls one or multiple resources is called a Resource Controller (RC). The RC can run directly on the resource (e.g. a PC), or run on a separate computer that can control the resources (e.g. a bunch of sensor nodes or an OpenFlow switch). OMF is designed to support many different types of resources, and additional RCs for those resources can make use of the FRCP APIs. All communication in OMF is done via Publish/Subscribe (PubSub). By default OMF uses AMQP for that, but other messaging layers such as XMPP are supported. Each OMF component must be able to talk to at least one PubSub server, although there can be multiple PubSub servers used in an experiment. The experimenter uses the Experiment Controller (EC) to run the experiment script and steer the resources. The EC can be installed on a user-facing machine inside the testbed, or alternatively on the user's own computer. The figure below shows the different components of the OMF6 architecture.

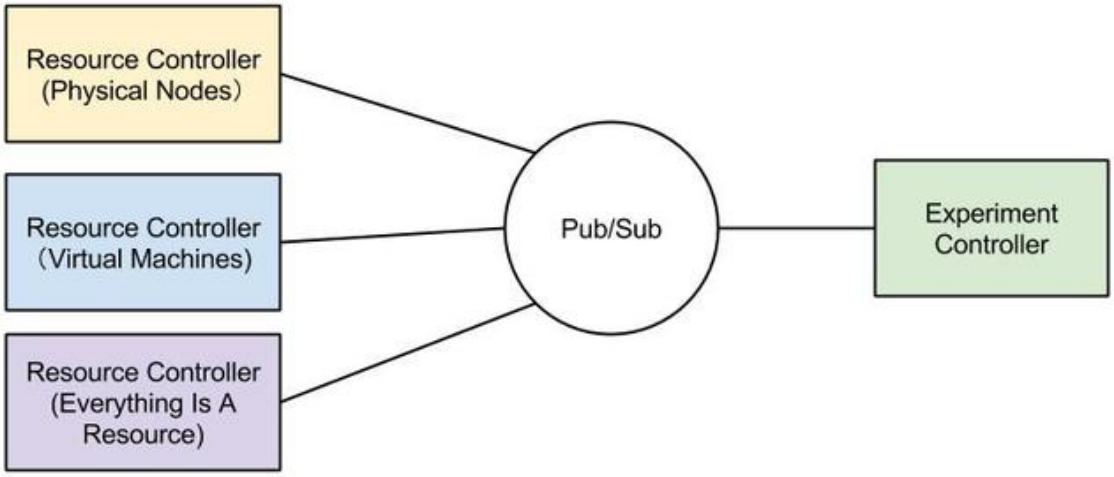


Figure 30 The OMF6 architecture.

OMF experiments are described in the OMF Experiment Description Language (OEDL [21]). OEDL is based on the Ruby programming language [22], but provides its own set of experiment-oriented commands and statements.

An OEDL experiment description is composed of 2 main parts. In the first part the resources are declared that will be used in the experiment, such as applications or computing nodes, and some configurations that the experimenter wants to apply to these resources. The second part defines the events that the experimenter would like to re-act during the experiment's execution, and the associated tasks that have to be executed when these events occur. A simple OMF example is included below.

```
defProperty("ap", "ap.sliceX.projectY.wilab2.ilabt.iminds.be", "Hostname of the access point")
defProperty("client", "client.sliceX.projectY.wilab2.ilabt.iminds.be", "Hostname of the client")

defApplication('ping') do |app|
  app.description = 'Simple Definition for the ping-oml2 application'
  app.binary_path = '/usr/bin/ping-oml2'
  app.defProperty('target', 'Address to ping', '', {:type => :string})
  app.defProperty('count', 'Number of times to ping', '-c', {:type => :integer})
end

defGroup('My_Pinger', property.client) do |g|
  g.addApplication("ping") do |app|
    app.setProperty('target', property.ap)
    app.setProperty('count', 3)
  end
end

onEvent(:ALL_UP) do |event|
  allGroups.startApplications
  after 5.seconds do
    Experiment.leave_memberships
    Experiment.done
  end
end
```

Figure 31 An example OMF OEDL script.

This script uses two properties named “ap” and “client” which contain the DNS names for the resources used in this experiment. The resource controllers on these resources are configured to subscribe to the AMQP server by using their hostname.

The defApplication code block describes the application that will be run on the resources. It contains a description and the location of the binary (on the resource). Optionally, it can contain some properties, which can be compared to command line arguments. In this case, a property is defined for the target of the ping and the number of times the ping has to run.

The defGroup code block describes the testbed resources that will be configured during the experiment. This example only interacts directly with the client resource. Therefore, the group ‘My_Pinger’ only contains one node, the client. Multiple resources can be added to a single group and even combinations of groups can be made. The content of the defGroup block maps the applications – defined in defApplication – to the resources and specifies a value for the properties.

The final part of the example contains the timeline of the experiment. After the ALL_UP event is received from all the involved resource controllers, the actual experiment starts and all applications are started in all groups (allGroups.startApplications), after which the actual ping command will be executed on the client. After 5 seconds (relative to the ALL_UP trigger), some cleanup actions are taken and the experiment finishes.

The example shown here is only a small excerpt of the capabilities of OMF. OMF also provides the ability to configure network interfaces of resources, execute events based on experiment measurement data or change properties of an application during the experiment execution. Recently a feature was added to retrieve the resources belonging to a certain slice.

To support the OMF framework in all WISHFUL testbeds, the following steps are foreseen:

- If the testbed hosts Linux-capable embedded PC’s, existing OMF Resource controllers can be installed to make the testbed FRCP compatible.
- For other testbeds, where resources are controlled through e.g. a server with a REST-API, the OMF RC can be installed on this server and not on the individual resources. This will be for example the case in testbeds with sensor nodes instead of embedded Linux PC’s.
- If some of the hardware in the WiSHFUL testbeds cannot be supported by OMF, new modules will be developed to plug into the resource controller.
- New OMF applications will be developed to support the tools described in chapter 0.

3.3 OMSP

The OML Measurement Stream Protocol [23] is used to describe and transport measurement tuples between Injection Points and Processing/Collection Points. All data injected in a Measurement Point (MP) is time stamped and sent to the destination as a Measurement Stream (MS). In the most common scenario, the measurement point is an application on a testbed resource and the collection point is represented by a database server where all measurements are stored. The liboml2 library (see 3.3.1) provides an API allowing to generate these measurement streams using this protocol and send them to a remote host (database server), or store them in a local file.

The usage of the OMSP can be split up in several phases:

- In the first step, a connection is made to a collection point (e.g. database server)
- After connection, a set of headers is sent, describing the injection point. The headers contain the following key/value parameters:
  - Protocol: OMSP version
  - Domain (or experiment-id): string identifying the experimental domain
  - Start-time: local UNIX time in seconds taken at the time the header is being sent

  - Sender-id: string identifying the source of this measurement stream
  - Application: string identifying the application producing the measurements
- Along with the headers, the schemas of the transported measurement streams are also sent to the collection point. Based upon these schemas, the collection point can then create the correct tables in the database.
- When the steps above have executed correctly, the measurement data (e.g. from a throughput testing application on a testbed resource) is time stamped and then serialised. The serialisation can either be either a binary or text encoding.

To every measurement sample some metadata is added:

- Timestamp: a double timestamp in seconds relative to the start-time sent in the headers, the server uses this information to rebase timestamps within its own timeline.
- Stream_id: an integer indicating which previously defined schema (see above) this sample follows.
- Seq_no: an int32 monotonically increasing sequence number in the context of this measurement stream.

The following data types are supported by OMSP: int32, uint32, int64, uint64, double, string, blob, guid, bool, guid. The latest version of OMSP also supports vector types, for which the type of its elements can be any of the data types listed above, except for string, blob or guid.

The OMSP architecture is shown in Figure 32.

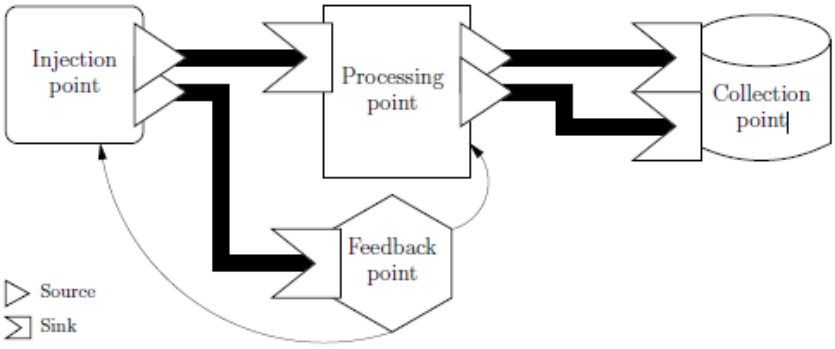


Figure 32 The OMSP architecture.

In the above example, an injection point creates two measurement streams. One of the MSs is handled by the processing point which creates two streams out of it and sends them to the collection point. The other stream originating at the injection point is sent to a feedback point which controls some aspect of the measurement tool at the injection point, as well as some of the processing point's parameters.

The next section shows an example client and server implementation for the OMSP protocol, named OML.

3.3.1 OML

OML [24] is an instrumentation tool that allows application writers to define customisable measurement points (MP) inside new or pre-existing applications. Experimenters running the applications can then direct the measurement streams (MS) from these MPs to remote collection points, for storage in measurement databases. OML was originally conceived to provide

measurement facilities for OMF-enabled testbeds, such as the Orbit testbed at RUTGERS. It is now a stand-alone tool which can also be run independent of OMF.

OML can instrument the whole software stack, and take input from any sensor with a software interface. It has no preconception on the type of software to be instrumented, nor does it force a specific data schema. Rather, it defines and implements a reporting protocol and a collection server. On the client side, any application can be instrumented using libraries abstracting the complexity of the network communication. Additionally, some of the libraries provide in-band filtering, allowing to adapt the measurement streams obtained from an instrumented application to the requirements of the current observation (e.g., time average rather than raw metrics). Applications for which the code is not available can also be integrated in the reporting chain by writing simple wrappers using one of the supported scripting languages (Python or Ruby). After collection from the distributed application, the time stamped data is stored in an SQL database (SQLite3 or PostgreSQL), grouped by experimental domain; a server can collect data from several domains at the same time.

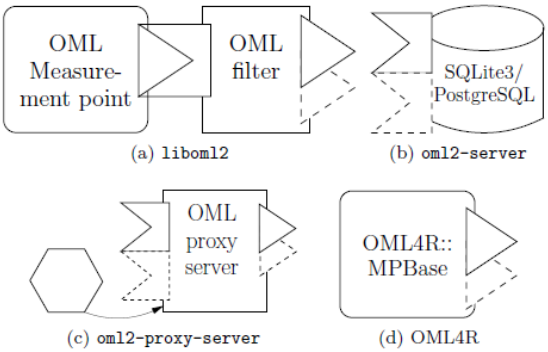


Figure 33 OML components.

OML consists of several components, which are shown in **Error! Reference source not found.**:

- *OML client library (liboml2):* the OML client library provides a C API for applications to collect measurements that they produce. The library includes a dynamically configurable filtering mechanism (OML filter) that can perform some processing on each measurement stream before it is forwarded to the *OML Server*. Next to the C library, there are also implementations available for Python (OML4Py) and Ruby (OML4R).
- *OML Server:* the OML server component is responsible for collecting and storing measurements inside a database. Currently, SQLite3 and PostgreSQL are supported as database backends.
- *OML proxy server:* this server allows the path between injection point and collection points to be decoupled. It also provides capabilities for disconnected experiments. It is currently used in the mobility framework in the w-iLab.t and will be used to cache measurement data in the portable testbed use case (see D6.1).
- *OML4R:* This is a native Ruby implementation for creating collection points. This is the most used implementation for applications that are not written in C and can thus not make direct use of the liboml2 C library.

A number of sample applications are available that perform measurements and filter and collect them using OML (e.g. iperf, libsigar and libtrace).

The use of OML in the Fed4FIRE architecture for measurement and monitoring is shown in the figure below.

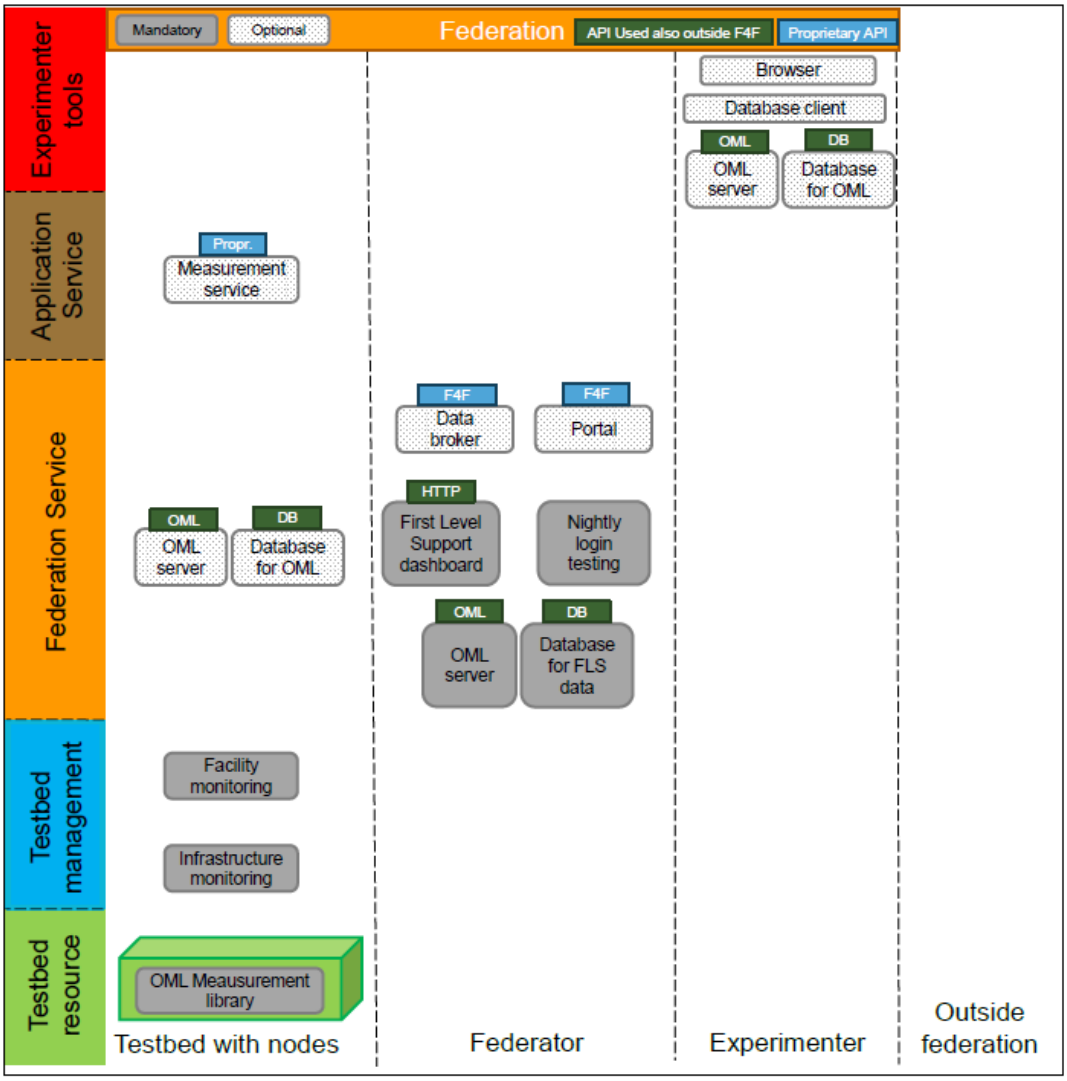


Figure 34 Fed4FIRE measurement and monitoring.

As depicted in Figure 34, OML consists of a service running on the testbed resource, and a service and database running on the OML server. On the resource, the Measurement Library (ML) takes measured values as an input, and is responsible for getting them added to the database at the OML server. Annotations to the measured values such are experiment ID, source ID and so on are automatically added by the OML framework. From an experimenter point of view, it is sufficient to redirect the measured value coming out of your own software or measurement tool to the ML to collect all of them in a single place for future processing. The experimenters can then use database tools (or even a browser) to query the measurement data. OML is also used in the F4F architecture to provide data for infrastructure and facility monitoring. OML adapters are available for proprietary measurement and monitoring services such as Zabbix [25] or Naggios [26].

OML will be used on WiSHFUL testbeds to collect both experiment measurement data and provide monitoring data for infrastructure and facility monitoring.

4 Development and F4F compatibility of new tools

This chapter describes some new tools that can be useful for experimenters of the WiSHFUL testbeds. Some of the tools described in the chapter below have functionality that is being re-used from projects like FP7 CREW. These tools will be redesigned so they comply with the Fed4FIRE standards and can thus be used in a uniform way through the F4F API's.

4.1 Advanced wireless monitoring tools

Two spectrum scanning tools are described in this chapter. The first scanning tool is a basic spectrum scanning tool using Wi-Spy devices. The second scanning tool is a more advanced version using a combination of different hardware devices in the w-iLab.t testbed. The tools presented here can be applied to any testbed, but installation of specific hardware might be required to provide the same functionality.

4.1.1 Basic spectrum scanning using Wi-Spy

The basic spectrum scanning is created to have an easily accessible tool (dubbed "spectro") for experimenters to have a quick and basic view on what the current channel occupation is on the w-iLabt.t Zwijnaarde testbed. The basic spectrum scanning tool is available at http://spectro.wilab2.ilabt.iminds.be , but an openVPN connection is required to access the page.

Using Wi-Spy sensing hardware[1] a database is built with power measurements over different channels. Depending on the sampling rate and frequency spectrum to be scanned, there are four modes available on the Wi-Spy DBx2 [27] used in w-ilab.t:

Table 1: Wi-Spy operation modes

Range ID	Name	Frequencies	Step (resolution)	Samples / sec
0	2.4GHz ISM	2400MHz-2483MHz	199.00KHz	419 samples
1	2.4GHz ISM FAST	2400MHz-2484MHz	560.00KHz	150 samples
2	5GHz	5100MHz-5866MHz	748.00KHz	1024 samples

[1] http://www.metageek.com/products/wi-spy

3	5GHz UN-II	5100MHz-5483MHz	374.00KHz	1024 samples

Every Wi-Spy device can be configured in one of the settings depicted in Table 1. Using, for example, setting #0, the device will scan the 2400 till 2483MHz band in increments of 199 KHz resulting in 419 samples every second.

These samples are sent to a database as illustrated in Figure 35.

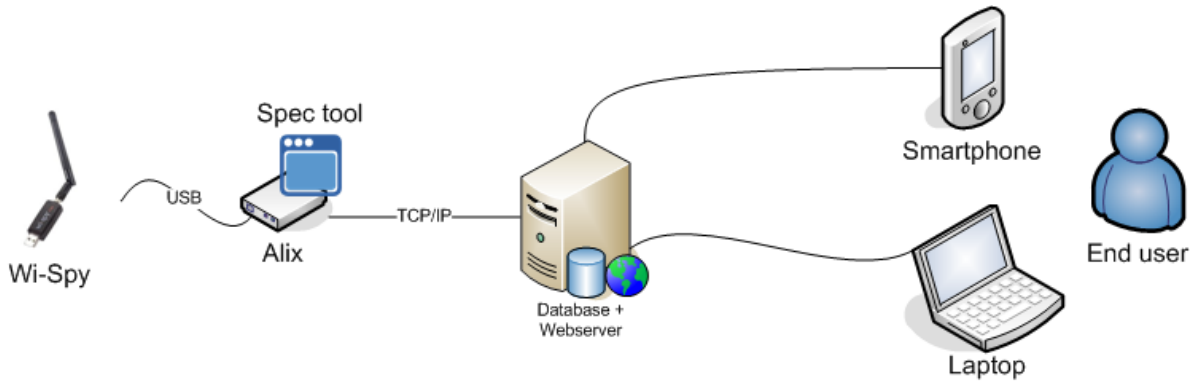


Figure 35: The Spectro architecture.

The Wi-Spy devices are connected via USB to an Alix embedded computer [28] (powered via PoE). This Alix serves as a forwarder: pushing all data the Wi-Spy collects on a TCP-socket. A small application that comes with the Wi-Spy toolbox, called *spectool* takes care of this. A small ruby script parses the output from the spectool and dumps the data into a database. Lastly, this database is queried by the webserver. The OML framework (see 3.3.1) is used for this purpose, so the resulting databases are compatible with the OMSP protocol (see 3.3) as used by Fed4FIRE.

When an end user connects to the database server (via a browser on a smartphone, laptop, computer, tablet, etc…), the data is fetched and drawn on a HTML5 canvas (illustrated in Figure 36). The horizontal axis has frequency labels, and the vertical axis has the timestamp of the measurement.

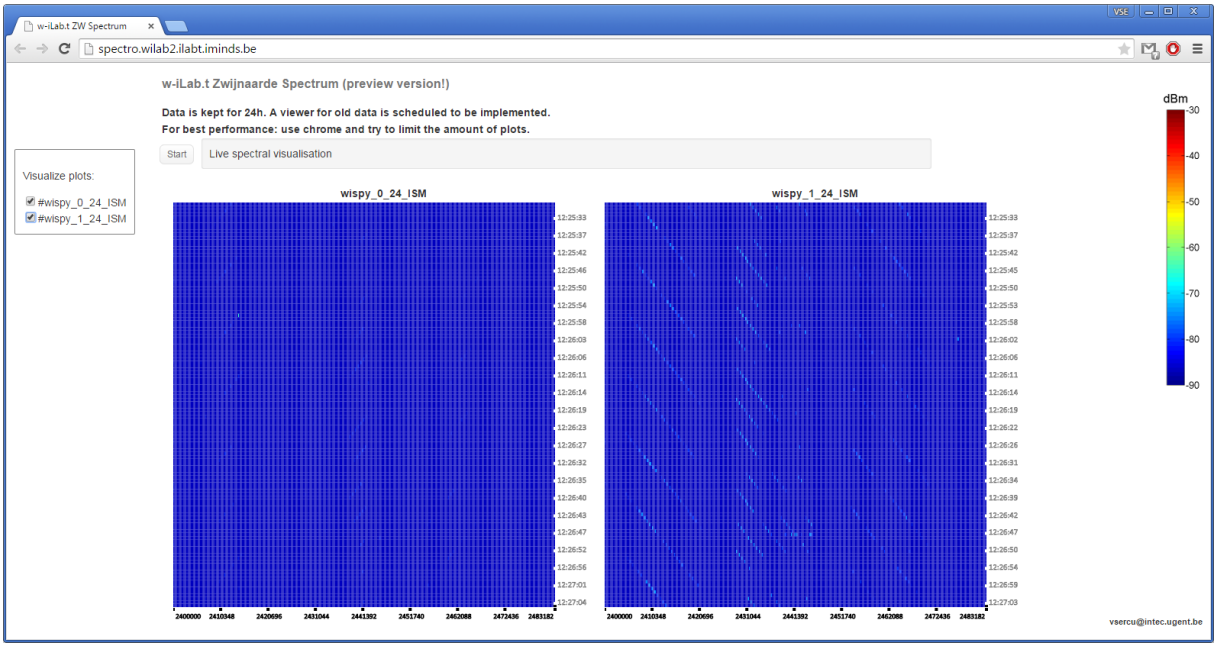


Figure 36: The webpage displaying the online spectrogram (Spectro).

Dataflow generated by the *spectool* is ASCII-text (one line per sample, RSSI-values that are separated by spaces) and looks like:

```
Wi-Spy DBx2 USB 2401370404: -56 -72 -85 -91 -95 -97 -100 -100 -102 -101 -102 -102 -
102 -102 -101 -102 -101 -102 -102 -102 -101 -101 -100 -102 -102 -102 -103 -102 -102
... omitted several samples ... -103 -102 -101 -102 -102 -103 -103 -103 -103 -103 -
103 -102 -103 -101 -103 -103 -102 -103 -104 -101 -103 -103 -103 -103 -102 -103 \n
```

The parsing done by the Ruby script (called an OML-wrapper script) will extract every RSSI-value and put it in the corresponding column, in right table. A database table is created for every measurement-set.

A PHP-script queries this data and allows it to be publicly viewed. Data queried from the database is available and is fetched using an AJAX mechanism that asynchronously fetches the RSSI-values stored in JSON format. The HTML5 canvas is filled will tiny squares that have colour values from red to blue (red being high RSSI values while blue represents low energy signals).

As the Wi-Spy devices scan continuously, data cannot be kept indefinitely. In the best case (Range ID#1, in Table 1), a single device generates 5KB (about the storage size of a row of 150 frequency measurements in the database) every second. Meaning 5KB * 60 (sec.) * 60 (min.) * 24 (hours) = 432 MB per day.

In order to cope with this, data is cycled every 24h for every Wi-Spy. Experimenters can query the database at any time and store the data locally if it is important for their tests.

Because of the very limited hardware requirements of this basic spectrum scanning solution, it can easily be used in other testbeds, or even in the deployment stage of the testbed-on-the-move (see D6.1). After installing the software on a laptop (Linux) and plugging in the Wi-Spy USB dongle, the experimenter can immediately get valuable information on the wireless spectrum at any location.

4.1.2 Distributed spectrum scanning

The distributed spectrum scanning tool was originally developed in the context of the FP7 CREW project. Its main goal is to allow experimenters to get a detailed view of the wireless spectrum before, during and after the execution of their experiment. The solution presented here will be made F4F compatible in the WiSHFUL project. This means that all resources required to do the scanning should be accessible through the Federation AM API, the experiment control should be done using an FRCP (3.2) compatible tool and the collection of the measurements has to be ensured by using an OMSP (3.3) compatible protocol. The experiment control is currently implemented in OMF5.4, so a conversion will be needed to OMF6 (3.2.1). For the collection of measurements, a similar version upgrade of the OML library (3.3.1) will be needed to ensure compatibility with the OMSP protocol.

The distributed spectrum scanning tool combines the information generated by several scanning components: USRP devices, Wi-Fi cards and imec [29] sensing engines. The 802.15.4 sensor nodes can also be used for spectrum scanning, but are not used in this solution. The general architecture of the tool is shown in the figure below. It consists of two parts: a web interface for visualization as the front-end, and a set of tools to monitor the wireless environment as the back-end. The figure below provides a general overview of the system with the different components. The different scanning components are described (and started) in an experiment description (OMF5.4). After starting the experiment, the measurements of the different scanning components are stored in the database and visualized on a web interface. The ZigBee monitor shown in the architectural figure will not be described here, since it only contains information about the number of packets (and their content) that are sent over a certain channel. A next version of the distributed spectrum scanning tool will include an integrated 802.15.4 sniffer that is capable of determining RSSI values on a certain frequency.

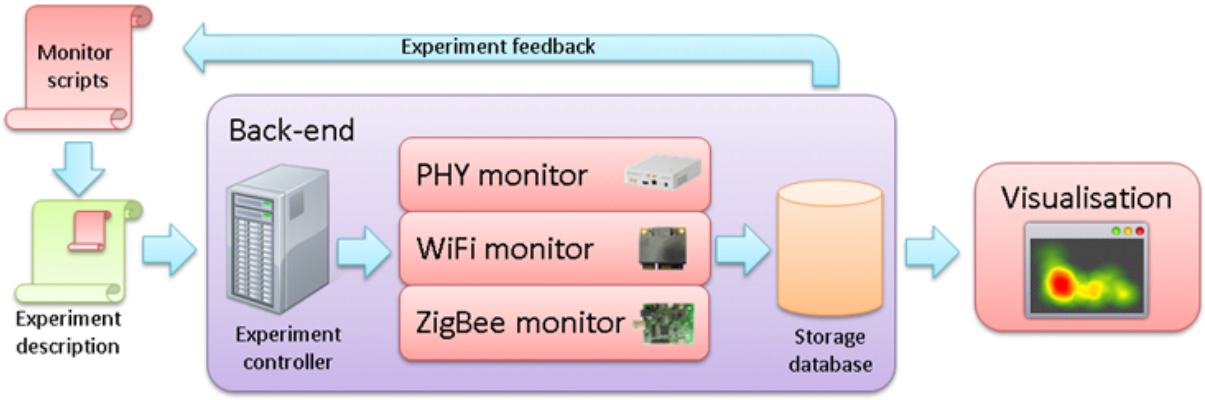


Figure 37 Distributed spectrum scanning tool.

The distributed spectrum scanning tool combines information that is gathered by the USRP, the imec sensing engine and finally by the Atheros Wi-Fi chipset.

The USRP devices can be used to obtain RSSI values of a certain frequency and are also able to determine the COR (Channel Occupancy Ratio). The frequencies in which the USRP can operate are limited to the frequencies in the ISM-band. The USRP enables researchers to rapidly design and implement powerful, flexible software radio systems. It offers high RF performance and great bandwidth. Next to spectrum monitoring, some of its functionality includes: physical layer prototyping, dynamic spectrum access and cognitive radio and networked sensor deployment.

The imec sensing engines are able to determine RSSI values inside and outside of the ISM-band. The device consists of two main blocks: an analogue RF front-end including analogue to digital conversion and a DIgital Front-end For Sensing (DIFFS). Two types of analog RF front-ends are available: a WARP board covering the 2.4 and 5 GHz ISM bands and an in house (imec) developed flexible SCAlable raDIO (SCALDIO), covering an RF input range from 0.1 up to 6 GHz and a channel bandwidth up to 40 MHz. The digital front-end is an ASIP specifically designed for sensing operations, signal conditioning and synchronization.

The more recent version of the Atheros [30] Wi-Fi chipset (version AR92xx and AR93xx or higher) have the built in capability of doing spectral analysis [31]. The chipset reports FFT (Fast Fourier Transform) data from the baseband under controlled conditions. The reports contain the following fields:

- The absolute magnitude for each FFT bin
- An index indicating the strongest FFT bin
- The maximum signal magnitude for each sample.

The information above can be used to create an open source spectrum analyser. The functionality was combined in a script called *ath_spec_scan*. The example below was tested on the wireless nodes in the w-iLab.t testbed using the ath9k wireless driver [32].

To get spectral information out of the Wi-Fi chipset, the devices have to be put in monitor mode. Prior to the scanning, the channel on which the user wants to scan has to be configured on the device. The output of the tool contains the following fields:

- current_time: Starting unix epoch timestamp (millisecond resolution)
- sniffer_mac: sniffer node Wi-Fi mac address
- frequency: working frequency (e.g. 2.412 GHz for 802.11g channel 1)
- COR: Channel Occupancy Ratio

- RSSI: Received Signal Strength indication
- previous_dur: measurement duration

The result of the distributed spectrum scan tool is a heat-map (background of Figure 38 and Figure 39) generated based on the RSSI of the entire 20 MHz that is monitored by the sensing devices (USRP, imec SE and Atheros chipset). The numbered circles in green represent embedded PCs with Atheros Wi-Fi cards. The purple circles represent imec sensing engines. The USRP devices are shown as smaller green circles, with numbers 63, 65, 69, 75, 81 and 89 (see Figure 39). Figure 38 shows the heat map constructed by combining the results of the Atheros chipsets and the imec sensing engines.

When clicking on individual nodes, a more detailed spectrogram will pop up, as shown in the figure below. Figure 38 shows the detailed spectrogram obtained at imec sensing engine 41. The spaces in between nodes are constructed by applying the inverse distance weighting algorithm to the measurement data.

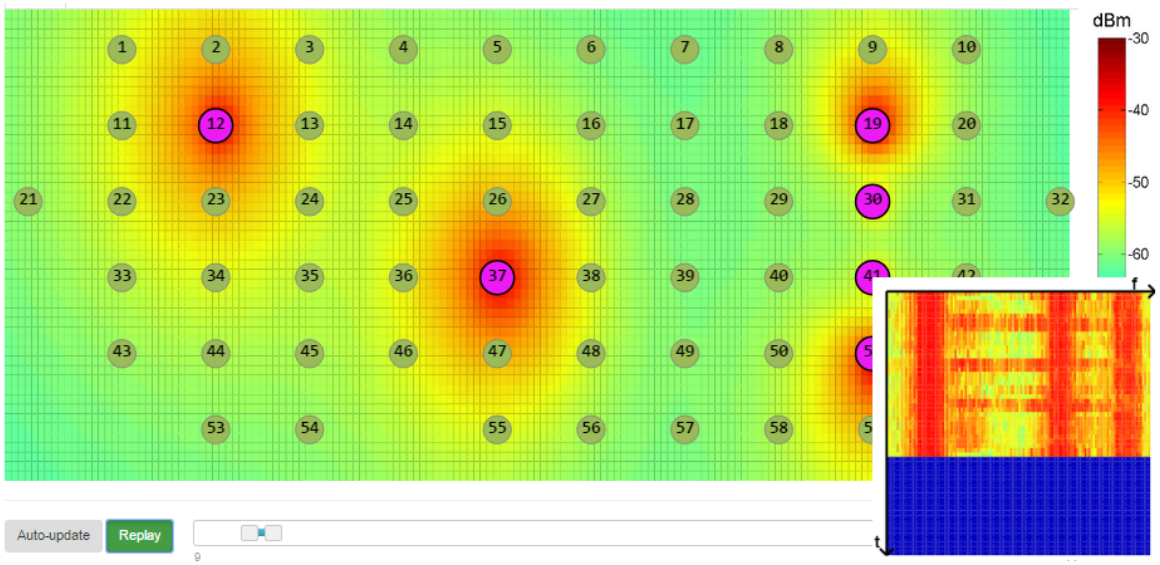


Figure 38 Spectrum scanning in w-iLab.t using an imec sensing engine.

Figure 39 shows the heat map constructed by the combination of the measurements taken by the Atheros chipsets and the USRP devices. It also shows the detailed spectrogram obtained by the Atheros chipset of Wi-Fi node 23.

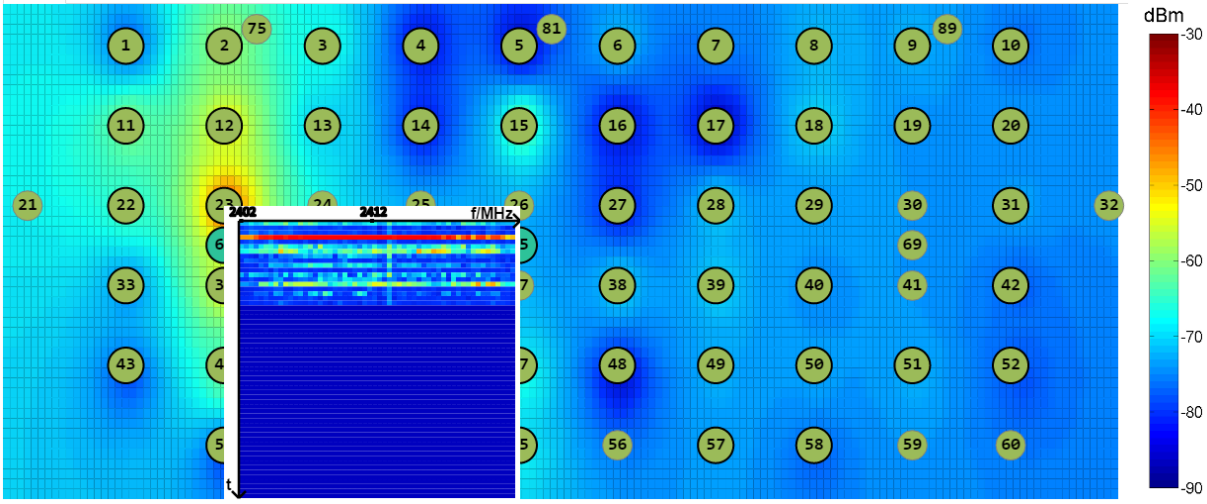


Figure 39 Spectrum scanning in w-iLab.t using an Atheros chipset.

4.2 Advanced tools for automation of experiments

4.2.1 Wireless experiment automation using the SUMO toolbox

Out of the box, the SUrrogate Modelling (SUMO) toolbox [33] is used as a complete multi-dimensional optimizer. It is targeted to achieve accurate models of a computationally intensive problem using reduced datasets. From the reduced datasets, the toolbox generates accurate Surrogate Models to evaluate the design objectives.

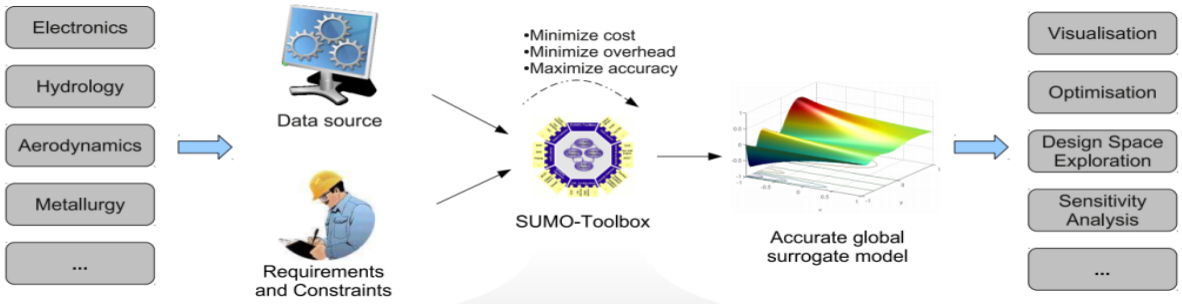


Figure 40 The process of generating an accurate surrogate model.

The SUMO toolbox bundles both the control and optimization functions together. The control function sitting at the highest level manages the optimization process with specific user inputs. The figure below describes the SUMO toolbox in a nutshell highlighting the control and optimization functions together.

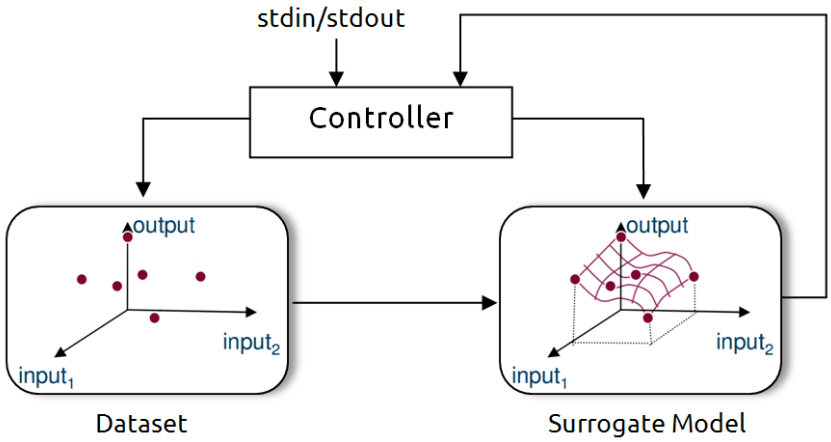


Figure 41 Out of the box SUMO toolbox in a nutshell view.

From the figure above it can be seen that the controller manages the optimization process starting from a given dataset (i.e. initial samples + output performance) and generates a surrogate model. The surrogate model approximates the dataset over the continuous design space range. Next, the controller predicts the next design space element from the constructed Surrogate model to further meet the optimization's objective. Depending on the user's configuration, the optimization process iterates until conditions are met.

Moreover, the aim of the SUMO toolbox is to use it as a standalone optimizer and put it inside an experimentation framework. This means from out of the box SUMO toolbox, the loop is broken, the control function is removed and clear input/output interfaces are created to interact with the controlling framework. The figure below shows the integration of the modified SUMO toolbox inside a wireless testbed.

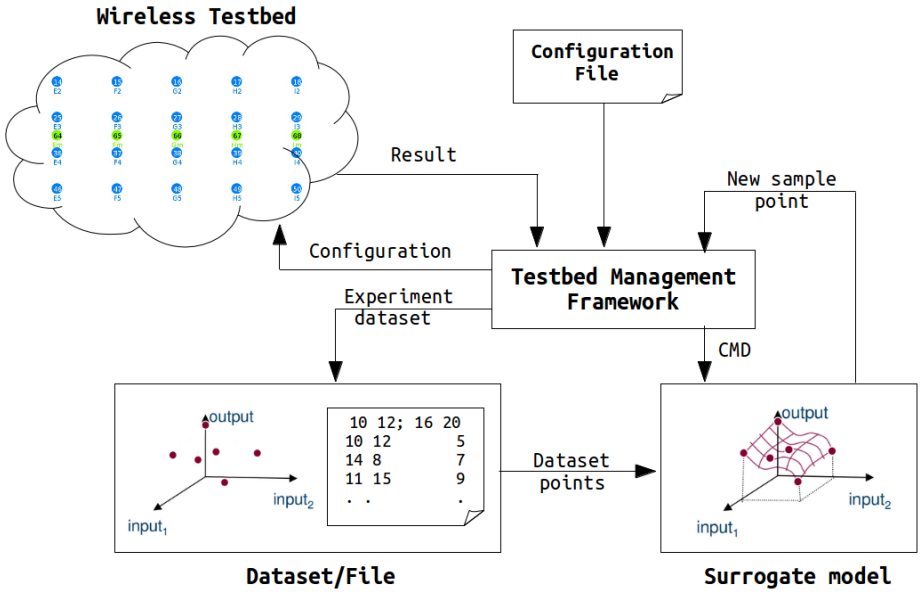


Figure 42 Integration of modified SUMO toolbox in a wireless testbed.

The figure above shows the use of the testbed management framework instead of the default controller. Suppose we are in a context of a wireless experiment, concerning two parameters: transmit power and the node location. The goal is to find the optimum combination of those two

parameters to achieve the maximum throughput. After the initial datasets are collected, SUMO toolbox determines the parameter set for the next experiment and writes the parameters in the dataset file, as shown in the figure above. The testbed framework performs the experiment and appends the result on the same row in the data file (in this case it is the throughput result). This iteration goes on until the stopping criteria are reached.

The testbed management framework performs the same tasks which were already implemented by the previous controller except additional tasks like experimentation on the wireless testbed, storing the dataset on a separate file, and reading experiment configuration from a file. It should be understood very well that the operation of the SUMO toolbox has not changed at all except replacement and addition of a few working blocks. A more general pictorial presentation on the operation of the SUMO toolbox is also presented in the next figure.

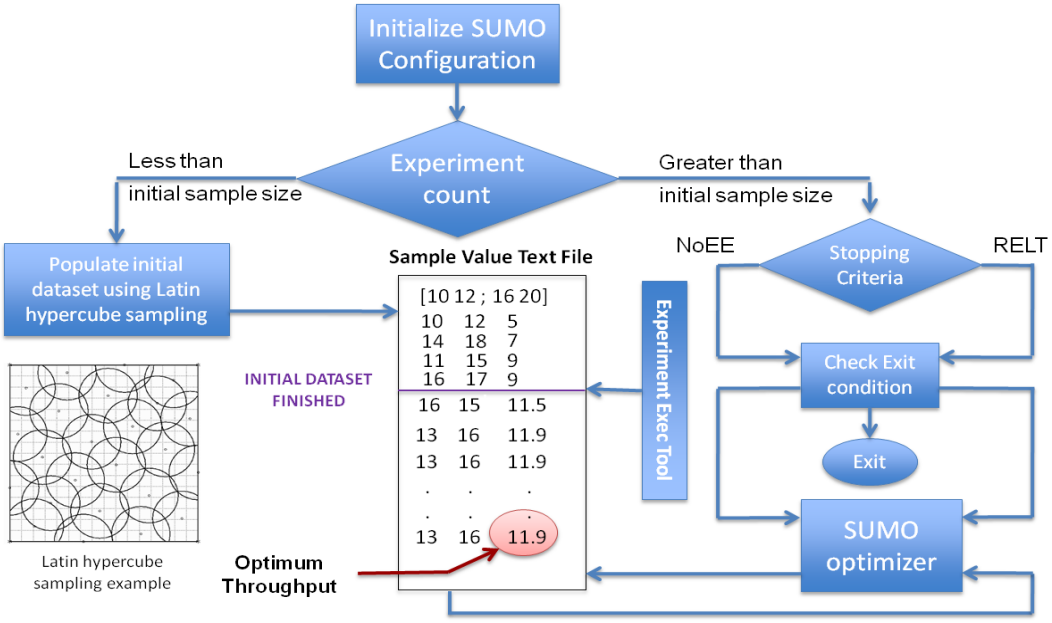


Figure 43 Complete modified SUMO toolbox optimization over two dimensional design space.

In the figure above, a complete optimization using the modified SUMO toolbox is presented. The SUMO toolbox optimizes a two dimensional design space (i.e. transmit power 10dbm to 16dbm and transmitter location id 12 to 20) problem.

Before SUMO toolbox starts the optimization process, it requires a minimum number of initial samples plus their output performance which will be used to generate the first surrogate model. The figure above shows four selected initial dataset pairs (i.e. [10 12], [14, 18], [11, 15], [16, 17]). Latin hypercube sampling is used to generate the initial samples, which guarantees the samples to be equally spaced across the design space.

Next, the experiment is conducted at each initial samples and the dataset (i.e. initial samples + output performance) is fed to the optimizer. The SUMO tool first generates the surrogate model and next calculates a new sample point to reach the global optimum. Using the calculated sample point, a new experiment is conducted and the dataset is updated. As the optimization progresses, the SUMO toolbox approaches the global optimum point. The figure below shows the different steps during the SUMO optimization process.

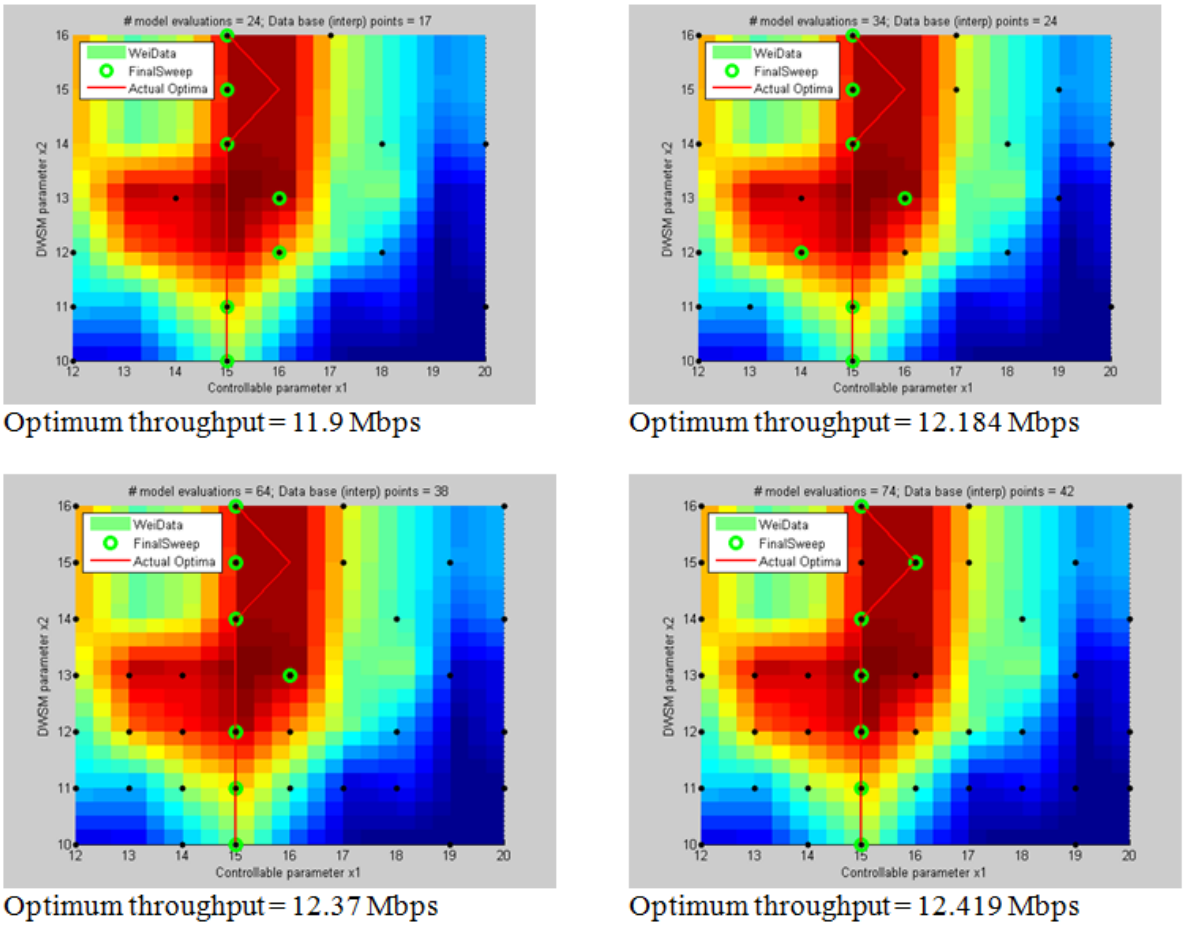


Figure 44 The different steps during SUMO optimization.

From the above figures, we see that as the number of iterations increases the simulation optimum coincides with the global optimum. The first figure shows 17 experiments and optimum performance of 11.9 Mbps. The second figure shows 24 experiments and optimum performance of 12.184 Mbps. The third figure shows 38 experiments and optimum performance of 12.37 Mbps. The last figure shows 38 experiments and optimum performance of 12.419 Mbps. At last, we end the optimization after a stopping criteria is met. Two stopping criteria widely used alongside the SUMO toolbox are when the Number of Experiment Equals (NoEE) a given value and when the Relative Error of performance is Less Than (RELT) a given threshold.

This tool will be modified so that it can be used by experimenters of the Fed4FIRE federation. Therefore all experiment control will be done using OMF6 (see 3.2.1).

4.3 Advanced tools for automation of testbed/software deployment

4.3.1 Ansible

Current testbeds are usually supported by diverse servers and require considerable amount of time and expert knowledge not only to install but also to maintain. This problem is particularly important for the portable testbed concept, where it should be possible to deploy a functional testbed with minimal effort and knowledge.

This problem has been also identified by IT administrators saying that "Deploying, configuring, and updating systems and applications can be drudgery". We can take their efforts in providing automation engines for application deployment to ease setting up a portable testbed in any location.

TUB is currently working with Ansible to make the testbed infrastructure configured through it. This gives an abstraction on what a testbed provider has to do to deploy the necessary software on a new machine. This also works with changing the configuration on one or multiple nodes. We are trying to follow the Ansible guidelines [34] to solve problems.

Ansible is an open source and free to use IT automation engine that aims for simplicity. It works by connecting to nodes and pushing out small programs, called “Ansible Modules” to them. These programs are written to be resource models of the desired state of the system. Ansible then executes these modules (over SSH by default), and removes them when finished. [35]

The library of modules can reside on any machine, and there are no servers, daemons, or databases required. Typically the administrator works with his favourite terminal program, a text editor, and probably a version control system to keep track of the changes to the content. [35]

Ansible can be compared to Chef (https://www.chef.io/chef/) or Puppet (https://puppetlabs.com/). Both of them use a client-server model with clients installed on all the servers you want to manage. This probably makes it more powerful, but also more complex than Ansible. It also means that they have higher demands on the controlled resources, while for Ansible the minimal requirement is only SSH based access. In later stage it uses Python for most of the work, but it still is not a strict requirement and Python itself can be deployed by Ansible.

Currently in the TWIST testbed we are managing internal DHCP/DNS server, webservices, XEN virtual machines, WLAN routers and robots (with workstation support). We have a central repository where all the configuration tasks are stored and are executed through Ansible playbooks. In practice it means that it is possible to bring a new machine that should control the robot, install the base system (ubuntu+ SSH server), add one line in the host inventory, and run a playbook. Based on all information in the playbook it will install all required packages that need to be on the robot, also prepare and compile robot environment. After that we have a fully functional machine for a robot support.

In the same manner, we are able to flash new image on the WLAN routers, install some additional packages (which are not installed by default due to flash size limitation, later we can use USB stick), and change any configuration we wish. All that is able to run in parallel to speed up the process, as a bonus a task that has already been executed once will not be executed second time (it still will be if there is some configuration mismatch).

Finally, for the portable testbed we envision this should help in creating a shortest path between a bare metal hardware with no software to a working testbed. Providing minimal requirements on the running software, namely SSH and networking, this can be an issue in a real portable testbed deployment, but not in the preparation phase. Everything else can be auto-deployed. Ansible can be particularly helpful when not all hardware is shipped by us or in preparation phase of the hardware for shipping. We have servers in mind here (data collection, management) but also particular wireless devices needing reconfiguration.

To summarize, we have found out that every hardware change requires a lot of manual installation process with expert knowledge to bring everything into a running state. We have decided to make an extra effort to do that process once in Ansible to simplify next changes. We already see benefits of that approach, as one can have a repository of all tools that have been deployed in the testbed and thus have a visible development process.

It is also possible to integrate Ansible into the FED4FIRE experiment life cycle. Namely the user activates some nodes using jFed (or another SFA) tool. This step loads an OS onto your nodes/servers and makes sure you can SSH to them. One of the activated nodes might be used as the Ansible terminal. Using Ansible, extra software can be installed like a web (or database) server, but also experiment control tools like OMF/OML. Especially the Ansible feature to use template files to fill in the correct values in configuration files gives much flexibility in experiment preparation and

customization. It could be used to properly configure the configuration files for the OMF resource controller/experiment controller. OML requires a database server and quite some configuration, so a playbook to install all of this will be useful. Usually a simple bash script is not enough for this as there are too many dependencies.

4.4 Tools for the integration of WiSHFUL UPIs in testbeds

The WiSHFUL UPIs developed in WP3 and WP4 will allow experimenters to implement control programs allowing fine-grained control of their solutions under test. The control programs can run on each node offering node-local control via UPI_R and UPI_N and management via the UPI_M. They can also enable network-wide or global control via UPI_G. This interface provides functions for discovering the nodes under control. Since this information is already available via the testbed management services, it makes sense to provide an interface between the testbed management services and the global monitoring and configuration engine implementing UPI_G. Figure 45 illustrates how this interface is integrated in the high-level WiSHFUL architecture.

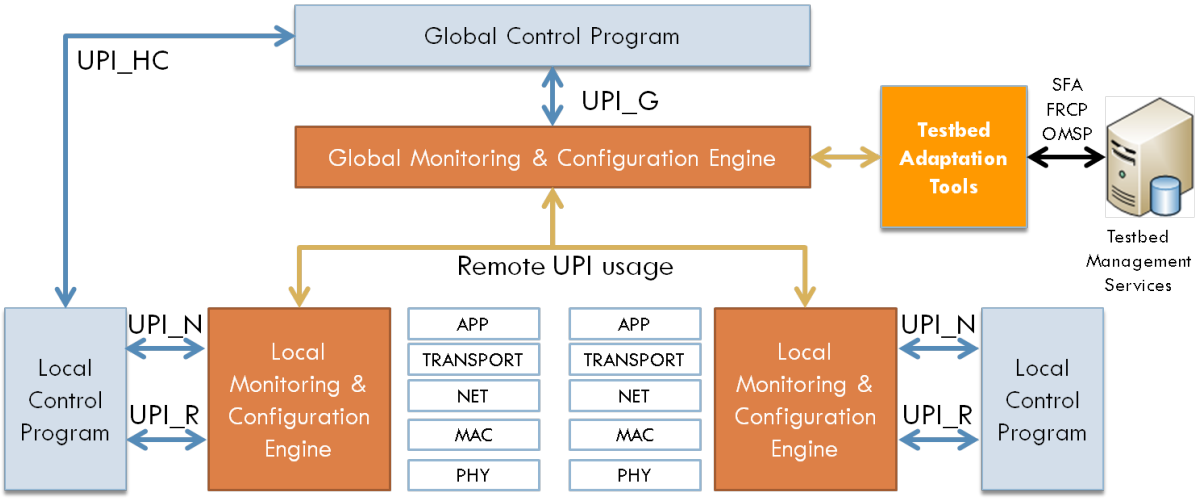


Figure 45 Testbed adaptation tools for integrating the WiSHFUL UPI's in testbed.

Because all Fed4FIRE compliant testbeds have a specific implementation (e.g. Rspec extensions) for the Fed4FIRE interfaces (SFA, FRCP, OMSP), testbed adaptation tools need to be developed that offer a generic interface to the global monitoring and configuration interface. SFA can be used for resource discovery, e.g. knowing which nodes are in an experiment. Using the resource descriptions advertised by the testbeds AM (in Rspec format) the type of nodes and their capabilities can be retrieved. FRCP allows configuring resources in an experiment, and is hence a possible candidate for implementing the protocol that allows remote UPI usage. Using slice information from the testbeds AM, also information about the groups defined in an experiment can be deducted from the AMQP topic subscription. This information is useful for experiments with multiple types of nodes (for instance mixed WiFi and ZigBee experiments). Additionally OMSP, provided by the OML library, can be used for gathering monitoring results from each testbed node. This information can be fed via UPI_G to the global control program to steer the decision making.

This section only describes the integration of WiSHFUL UPIs at a very high level. It will be described in detail in D5.2 at month 12.

5 Conclusion

In this deliverable, we described the Fed4FIRE compliance status of all WiSHFUL testbeds and their implementation efforts to join the federation. The TWIST and IRIS testbeds make use of the Geni

Control Framework to expose the federation AM API. The SFAwrap tool is used to bring the FIBRE island at UFRJ up to Fed4FIRE standards. To better understand the SFAwrap and GCF tools, a description of the SFA architecture is included in this deliverable. Finally, the Orbit and w-iLab.t are already F4F compliant. Their architecture is included in this deliverable to have a complete list of all WiSHFUL testbeds.

Providing a federation AM API on all testbeds is the first step in the federation. Once the federation AM API's are up and running, several client side federation tools are offered by Fed4FIRE. jFed is chosen as the tool to implement the SFA protocol and provide experimenters with the functionality to discover, reserve and provision resources in all WiSHFUL testbeds. To support the FRCP protocol and the associated advanced experiment control functionality, the OMF6 tools were chosen. Finally, the OML library implements the OMSP protocol and provides an easy way for experimenters and facility providers to collect measurement and monitoring data. The above tools will be extended to support the hardware and software in WiSHFUL, starting with the jFed tool, since this tool is used for the first stage in the experiment life cycle.

Finally, some new tools are presented and modified to support the Fed4FIRE federation protocols like SFA, FRCP and OML. The tools offer functionality that facilitates the experimentation process for users of WiSHFUL infrastructure, including users of the portable testbed. The tools presented here are spanning advanced wireless spectrum monitoring, automation of experiments and automation of software deployment. To conclude, a short description is given on the integration of WiSHFUL UPIs into a testbed environment. The need for other tools will be investigated during the course of the project. Feedback from open call experiments and/or extensions will play a crucial role in deciding what tools are useful for users of WiSHFUL.

6 References

[1] Slice Federation Architecture 2.0 (available at http://groups.geni.net/geni/wiki/SliceFedArch and https://fed4fire-testbeds.ilabt.iminds.be/asciidoc/general.html)

[2] Clearinghouse (available at http://groups.geni.net/geni/wiki/GeniClearinghouse)

[3] Description of the federation AM API (available at https://fed4fire-testbeds.ilabt.iminds.be/asciidoc/federation-am-api.html)

[4] GENI, Global Environment for Network Innovations (available at https://www.geni.net/)

[5] RSpecs, Resource Specification documents (see https://fed4fire-testbeds.ilabt.iminds.be/asciidoc/rspec.html)

[6] GENI Control Framework (available at http://trac.gpolab.bbn.com/gcf)

[7] GENI-tools (available at https://github.com/GENI-NSF/geni-tools/wiki)

[8] USRP N210 series (datasheet available at http://www.ettus.com/content/files/07495_Ettus_N200-210_DS_Flyer_HR_1.pdf)

[9] SFAwra.p (available at http://sfawrap.info/)

[10] MySlice portal (available at https://myslice.info/)

[11] W-iLab.t wireless testbed (http://doc.ilabt.iminds.be/ilabt-documentation/wilabfacility.html)

[12] Emulab – Network Emulation Testbed software (available at http://emulab.net/)

[13] GENI Aggregate Manager API Version 3 (available at http://groups.geni.net/geni/wiki/GAPI_AM_API_V3)

[14] jFed, a Java-based framework for testbed federation (available at http://jfed.iminds.be/)

[15] XEN Hypervisor (available at http://wiki.xen.org/wiki/Xen_Project_Software_Overview)

[16] OpenVZ, container based virtualization for Linux (available at https://openvz.org/Main_Page)

[17] FRCP, Federated Resource Control Protocol (available at https://github.com/mytestbed/specification/blob/master/FRCP.md)

[18] XMPP, eXtensible Messaging and Presence Protocol (available at http://xmpp.org/)

[19] AMQP, Advanced Message Queuing Protocol (available at https://www.amqp.org/)

[20] OMF6, cOntrol and Management Framework (available at https://omf.mytestbed.net/projects/omf6/wiki/Wiki)

[21] OEDL, the OMF Experiment Description Language (available at https://omf.mytestbed.net/projects/omf6/wiki/Wiki)

[22] Ruby Programming Language (available at https://www.ruby-lang.org/en/)

[23] OMSP, The OML Measurement Stream Protocol (available at http://oml.mytestbed.net/doc/oml/2.11/doxygen/omsp.html)

[24] OML, (Orbit) Measurement Library (available at http://oml.mytestbed.net/)

[25] Zabbix monitoring solutions (available at http://www.zabbix.com/)

[26] Nagios monitoring solutions (available at http://www.nagios.org/)

[27] Wi-Spy spectrum monitoring tool (available at http://www.metageek.com/products/wi-spy)

[28] Alix embedded PC's (available at http://www.pcengines.ch/alix.htm)

[29] Imec research centre (available at http://www2.imec.be/be_en/research.html)

[30] Atheros Wi-Fi chipsets (available at http://www.qca.qualcomm.com/)

[31] Atheros spectral scan (available at https://wireless.wiki.kernel.org/en/users/drivers/ath9k/spectral_scan)

[32] Ath9k Wireless Driver (available at https://wireless.wiki.kernel.org/en/users/drivers/ath9k)

[33] Surrogate Modeling toolbox (available at http://sumowiki.intec.ugent.be)

[34] Ansible playbooks (available at http://docs.ansible.com/playbooks.html)

[35] Ansible architecture (available at http://www.ansible.com/how-ansible-works)