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

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

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

Deliverable D4.1 FELIX Components Validation Report

Version 1.0

Due date:	31/08/2015
Submission date:	01/09/2015
Deliverable leader:	i2CAT
Author list:	Carlos Bermudo (i2CAT), Carolina Fernandez (i2CAT), Bartosz Belter (PSNC), Krzysztof Dombek (PSNC), Artur Juszczyk (PSNC), Kostas Pentikousis (EICT), Umar Toseef (EICT), Gino Carrozzo (NXW), Roberto Monno (NXW), Atsuko Takefusa (AIST), Jason Haga (AIST), Tomohiro Kudoh (AIST), Takatoshi Ikeda (KDDI), Jin Tanaka (KDDI), Brecht Vermeulen (iMinds), Vicent Borja Torres (iMinds)

Dissemination level

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

<THIS PAGE IS INTENTIONALLY LEFT BLANK>

Table of Contents

List of Figures

List of Tables

Abstract

This report documents the procedures used to validate and test the internal workflows, on the one hand, and interactions between the FELIX components, on the other. The document attempts to effectively demonstrates that the implementation exhibits the expected behaviour as a whole. The document also sums up the steps carried out to deploy the FELIX framework software modules across the federated FELIX islands. Finally, this document describes the status of the current deployments and indicates some of the missing steps that are to be addressed during the final year.

Excecutive Summary

Deliverable D4.1 details the deployment procedures, functional testing and validation, and the evaluation of the validation for each of the FELIX components.

Validation is not an isolated step from the development process; it is rather performed throughout the development and deployment phases of each module providing feedback to solve and improve them in a cyclic way.

The document is structured in three main parts: the deployment of the components, the testing of functionalities and the validation tools. After a brief introduction and a list of acronyms and short definitions of the concepts used in the document, the deployment of components section provides details on where each component has been installed. After that, the Testing of functionalities section describes per component which features are validated and how. Finally, the Validation Tools section provides a list of the different tools used to validate the components (OMNI, Jenkins with SonarQube, etc.). As these tools have been presented and described in previous deliverables, this document will briefly describe them but focus on how we used them to perform different kind of validations on the software modules.

This document is addressed to software architects, software engineers and software developers implementing specific features of FELIX to know how to validate future development or improvements.

1 Introduction

In software product development, the process of validating the software product is aimed at ensuring that the design requirements initially defined in the architecture specifications are met and properly implemented by the product. The **validation** process usually takes place at the end of developments, once *verification* has been completed. Whilst the verification process (generally consisting in functional testing) ensures that the software module has been developed to comply with the initial design requirements and specifications, the *validation* stage checks that the software module actually meets the needs defined by such design requirements. The *verification* and *validation* processes allow to identify unexpected component errors or faults, and in this case, result in feedback to the development team to fix a specific issue.

In FELIX, the validation and verification processes have been carried out by two different groups:

- Developers of each software module run the functional testing or verification, both during development and after the end of the developing stage
- Integrators (island owners) execute the software validation, mostly related to deploying FELIX components in the target islands and implementing use cases.

The work distribution described above responds to needs to perform different types of test in different stages of maturity of the software modules.

During and after the development stage, FELIX developers teams have performed automatic and/or manual tests on their software module to verify proper internal functioning, as well as interaction with external modules. Once a software module is extended with new functionalities, further tests are carried out to prepare and ease future integration with a subset of modules. Such modules are specific for every type of FELIX software module. When the integration stage starts, the integration team takes over the testing role and shall thus provide further feedback to the developers team, in case any issue was encountered on the software module.

2 Abbreviations and Definitions

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

2.1 Abbreviations

- **AAA**: Authentication, Authorisation and Accountability.
- **CLI**: Command Line Interface.
- **CRL**: Certificate Revocation List.
- **CRM**: Computing Resource Manager.
- **F4F-FLS**: Fed4FIRE's First Level Support Monitoring.
- **GENI**: Global Environment for Network Innovations.
- **GUI**: Graphical User Interface.
- **MMS**: Master Monitoring System.
- **MRO**: Master Resource Orchestrator.
- **MS**: Monitoring System.
- **OFVER**: OFELIA VERsioning system.
- **PE**: Policy Engine.
- **pyPElib**: python Policy Engine library.
- **RM**: Resource Manager.
- **RO**: Resource Orchestrator.
- **RSpec**: Resource Specification.
- **SDNRM**: Software Defined Network Resource Manager.
- **SERM**: Stitching entity Resource Manager.
- **STP**: Service Termination Point.
- **TNRM**: Transit Network Resource Manager
- **URN**: Uniform Resource Name.
- **VM**: Virtual Machine.

2.2 Definitions

- **Agent**: Refers to the virtualisation server. This is the software running in each virtualisation server and acting as the entry point to the hypervisor that allows to manage the virtual machines of the users.
- **cURL**: Library and command-line tool for transferring data using various protocols.
- **FlowSpace**: Set of rules to define operations on packets. Contains a variable number of datapath IDs and their selected ports, a filtering condition to match the packets (usually a VLAN or a range of them). This conforms an internal data model of the FlowVisor that is later on inserted on the switches.
- **GENI**: Provides a virtual laboratory for networking and distributed systems research and education, as well as fostering standardisation and making the SFA interfaces advance.
- **Island**: Physical domain under particular management. It provides infrastructure and resources to the end user.
- **OFVER**: Versioning system that consists of a number of core scripts to manage the install and update processes, and allows extension through custom scripts.
- **OMNI**: CLI tool which is part of the GENI Control Framework.
- **perfSONAR**: a network measurement toolkit that provides, among others, a uniform interface to schedule measurements and retrieve data from the network devices.
- **pyPElib**: Policy Engine library developed in Python. It aims to help programmers using the abstractions provided to apply rule-based policy enforcement.
- **Pylint**: Python source code analyser which looks for programming errors, helps enforcing a coding standard and sniffs for some improper programming practices.
- **RM**: Software component able to reserve, create, manage and delete resources by communicating with the hardware. It provides interfaces for both administrative and common operations on resources.
- **RSpec**: XML document following agreed schemas to represent resources that are understood by Resource and Aggregate Managers.
- **URN**: Public identifiers given to resources in the network in order to uniquely identify and exhaustively describe the properties of the resource. For that, the *urn* scheme is followed.
- **Zabbix**: Production-grade software to perform real-time monitoring of metrics collected from servers, virtual machines and network devices.

3 Deployment of Components

The deployment of the FELIX stack has followed, at first, a sequential planning of different modules within an island in order to validate the behaviour of the modules and to generate and improve the deployment instructions required to simplify installation and configuration in other domains. Thereafter, FELIX SW modules started being deployed in the multiple domains, where it took place an iterative refinement of the configurations and integration steps for the different modules. With most modules deployed, the basic communication workflows between them were set in place and validated.

Figure 3.1 depicts the location of each domain containing FELIX software modules, on top of a world a map. Below, a table is shown containing the status of deployment and validation of the user's GENIv3 north-bound APIs to reach every module.

A brief explanation of the deployment steps for the different modules is provided below, aimed at integrating the live representation of islands composition and software status maintained live on the project website (`http://www.ict-felix.eu/?page_id=332`). Appendix A provides a number of tables consisting on a detailed summary of the deployment status and information of each component per island or domain. This information is shared among the FELIX domains in order to properly configure the Resource Orchestrators and Master Resource Orchestrators within the federated testbeds.

3.1 Resource Orchestrator

The Resource Orchestrator (RO) is at the heart of the FELIX virtual infrastructure configuration, coordinating user's actions for resource instantiation (computing, OpenFlow-enabled switches, stitching entities and transit networks) with C-BAS and monitoring. RO can run in two operation modes: RO (default) and Master RO (MRO), thus acting as parent orchestrator and coordinator of other ROs.

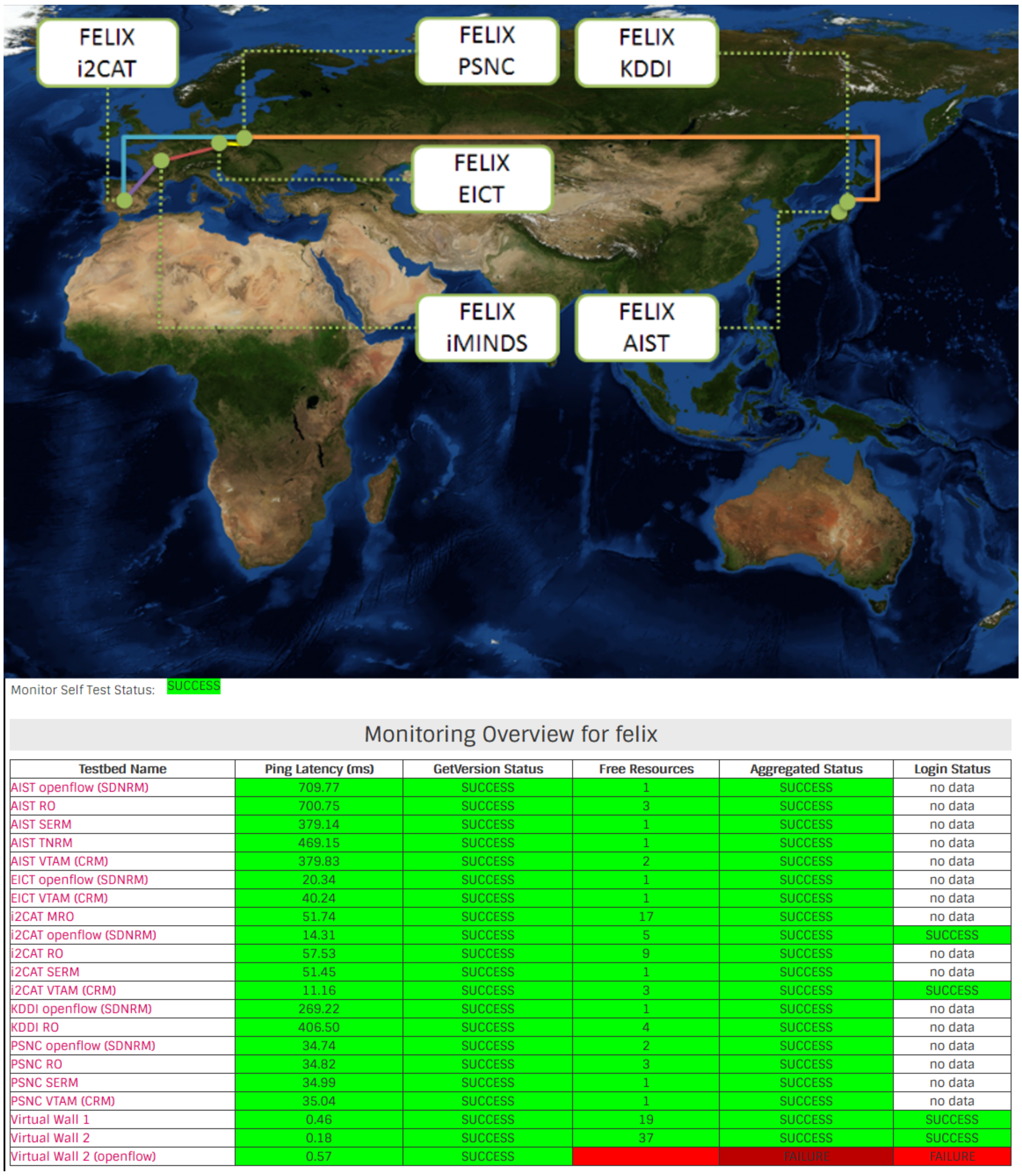
Each island (but iMinds, where FOAM is reused from Fed4FIRE) has deployed its own RO, which is responsible for orchestrating local resources made available for the federation. As per the design of the FELIX architecture, two MROs are deployed in FELIX: one in i2CAT (European region) and one in AIST (Japanese region). MROs communicate with each other to implement inter-continental infrastructure provisioning. Further details on the location of such modules are provided in Appendix A.

The source code for RO is available in the GitHub public repository, under the **resource-orchestrator** branch. A number of configuration files are provided so that the administrator fills those with data such as the address, port and endpoint provided per RO (see "Annex A" to consult them) or enabling/disabling features during testing phases, such as *RSpec* validation, credential verification and so on. After introducing the Monitoring System (MS), we saw necessary to add extra configuration files for CRM, SDNRM and SERM in order to provide the missing information on how to access monitoring information (i.e. metrics) on the hardware. This information is not provided by the GENIv3 APIs, therefore is filled internally by each administrator during the configuration of RO/MRO. o help with this process, a sample configuration file is provided with blanks to be filled.

The internal communications of RO-MRO have been interactively tested and validated to consolidate the intra-region orchestration workflows and GENIv3 API contents described in WP3 deliverables, e.g. towards the resource database, the various RMs or the northbound API to users.

3.2 SDNRM

SDNRM provides users with the possibility of defining a number of flows (*traffic rules*) over a subset of physical devices, and the administrators with the ability of granting and provisioning them. This component, coming from previous FP7 projects and extended by i2CAT, was deployed in almost every FELIX island; namely i2CAT, PSNC, EICT, KDDI and AIST islands. In the case of the iMinds island, the FOAM module (a component with similar functionalities as SDNRM) was already deployed and thus used in that island.



Monitor Self Test Status: SUCCESS

Monitoring Overview for felix

Testbed Name	Ping Latency (ms)	GetVersion Status	Free Resources	Aggregated Status	Login Status
AIST openflow (SDNRM)	709.77	SUCCESS	1	SUCCESS	no data
AIST RO	700.75	SUCCESS	3	SUCCESS	no data
AIST SERM	379.14	SUCCESS	1	SUCCESS	no data
AIST TNRM	469.15	SUCCESS	1	SUCCESS	no data
AIST VTAM (CRM)	379.83	SUCCESS	2	SUCCESS	no data
EICT openflow (SDNRM)	20.34	SUCCESS	1	SUCCESS	no data
EICT VTAM (CRM)	40.24	SUCCESS	1	SUCCESS	no data
i2CAT MRO	51.74	SUCCESS	17	SUCCESS	no data
i2CAT openflow (SDNRM)	14.31	SUCCESS	5	SUCCESS	SUCCESS
i2CAT RO	57.53	SUCCESS	9	SUCCESS	no data
i2CAT SERM	51.45	SUCCESS	1	SUCCESS	no data
i2CAT VTAM (CRM)	11.16	SUCCESS	3	SUCCESS	SUCCESS
KDDI openflow (SDNRM)	269.22	SUCCESS	1	SUCCESS	no data
KDDI RO	406.50	SUCCESS	4	SUCCESS	no data
PSNC openflow (SDNRM)	34.74	SUCCESS	2	SUCCESS	no data
PSNC RO	34.82	SUCCESS	3	SUCCESS	no data
PSNC SERM	34.99	SUCCESS	1	SUCCESS	no data
PSNC VTAM (CRM)	35.04	SUCCESS	1	SUCCESS	no data
Virtual Wall 1	0.46	SUCCESS	19	SUCCESS	SUCCESS
Virtual Wall 2	0.18	SUCCESS	37	SUCCESS	SUCCESS
Virtual Wall 2 (openflow)	0.57	SUCCESS	🟥	FAILURE	FAILURE

Figure 3.1: Deployment of components in FELIX islands

The source of this component is available under the **ocf** branch (*optin_manager* folder) and instructions are provided in the corresponding section of the GitHub public wiki page. In the source, a sample configuration Python file (*localsettings-example*) is provided so that it can be copied and filled by the island administrator into a new file (*localsettings.py*), containing details such as the access for the MySQL database, administrator e-mail for notifications and so on.

The second step of configuration requires the installation of the module in the server and accessing to the administrator GUI to set up connection to the FlowVisor slicing SDN controller that operates in each FELIX island.

As the RO performs an intra-island monitoring process (i.e. retrieve SNMP access information for its managed SDN-enabled HW equipment), a new sample configuration file (*sdnrm.json.example*) was added in the **resource-orchestrator** branch to include such non-public information. This file must be copied as a new file (*sdnrm.json*) and filled with the URN of each device and its IP, SNMP port and community string. Such information is to be later provided to the (M)MS for internal use.

Finally, the information provided in the mentioned JSON file (plus a list of the ports to be monitored) is also provided within the topology definition XML file of the SNMP manager, as part of another configuration step that allows the perfSONAR Sequel service to retrieve metrics from the switches. Such components are third-party tools that do not belong in MS or MMS, and are currently deployed in i2CAT and PSNC domains to provide metric information to MS.

3.3 CRM

CRM is the software module that interfaces with the virtualisation hardware available at each domain so as to create virtual machines. This component was developed fully by i2CAT and brought as well from previous FP7 projects. Within FELIX, it was subsequently extended by i2CAT and also by AIST, the latter adding interfaces and developing required extensions to allow creation of VMs on top of KVM-enabled servers.

Therefore, there are two flavours, according to the equipment provided by the domain of each organisation: *XEN-CRM* (the original version, deployed by i2CAT, PSNC and EICT) and *KVM-CRM* (the version extended by AIST, currently deployed in AIST and to be placed in KDDI as well). The case of iMinds is a bit different, as its infrastructure runs with VirtualWall, an alternative to the combined CRM and SERM. In this domain, thus, they use VirtualWall along with its implicit stitching technique.

The source of XEN-CRM is available under the **ocf** branch (*vt_manager* folder) and instructions are provided in the corresponding section of the GitHub public wiki page. In a similar way as SDNRM, both flavours of CRM provide a sample configuration Python file (*mySettings-example*) to be later copied and filled by the island administrator into a new file (*mySettings.py*). That file will contain details such as the access for the MySQL database, administrator e-mail for notifications and so on.

The second step of configuration requires the installation of the module in the server and accessing the administrator GUI to set up the ranges of IP and MAC addresses that are assigned under her domain, as well as basic information about each server machine to be used for VM provisioning (e.g. name, OS, and more importantly, the location (that is, address and port) of the agent software to send petitions, bridge interfaces and connection of each server with the OpenFlow-enabled switches. This information is, among others, appropriately translated from the physical, real configuration, into the network logical conformation that the users are aware of and which is of use for their experiments. Specifically, the details provided to the experimenters contain which interface of their VMs are connected to every switch, and are provided to them via the GENIv3 API (i.e. a *ListResources* call).

As with SDNRM, and SERM, a new sample configuration file (*crm.json.example*) was added in the **resource-orchestrator** branch to include the information required for monitoring access, which is not provided by the GENIv3 *ListResources* call. This file must be copied as a new file (*crm.json*) and filled with the same data and purpose as for SDNRM.

3.4 SERM

SERM's functionality allows to send user traffic between islands via different types of transport networks (i.e.: static/dynamic VLAN-based network provider services or GRE tunnels via Internet) and thus it must be deployed in every FELIX island.

The SERM module, developed by PSNC, was deployed within the PSNC, i2CAT, KDDI and AIST islands. The source code is available in the FELIX GitHub public repository, at the **stitching-entity** branch. The installation of dependencies and the deployment of the SERM is performed in a way similar to that of the RO, as it follows a similar structure and scripts for deployment.

There are two options for the configuration and deployment of the SERM component, which differ on the way the stitching switch is controlled:

- SERM component interacting with Ryu [1]: this option was implemented in PSNC, KDDI and AIST islands where a Ryu OpenFlow controller was set to run in order to manage the OpenFlow switch that was used as the data plane stitching element.
- SERM component interacting with POX [2]: this option was implemented in the i2CAT island, as there is a POX OpenFlow controller that manages the OpenFlow switch to be used as the data plane stitching element.

In each island deploying SERM component, a configuration file (*se-config.yaml*) is to be filled by the domain administrator with the information on stitching switch ports and related remote ports located in different islands, as well as information related to TNRM's functionality (e.g. type of transport network, VLAN range, etc.). In the YAML configuration file, it is also declared whether the SERM module communicates with the Ryu or POX controller in order to control the stitching switch.

The initial deployment of the SERM component was also carried out in EICT, but due to some differences in its network topology (i.e. no NSI inter-connections are provided), the deployment process was held until the TNRM supports GRE tunnels; when that module will be deployed. A different approach was taken in case of iMinds island. As iMinds' VirtualWall (somewhat equivalent to CRM) provides an integrated SERM functionality within their VirtualWall environment, the SERM component was not deployed there.

3.5 TNRM

The Transit Network Resource Manager (TNRM) is one of the special software components within the FELIX architecture, since it is not distributed over the different domains in FELIX. Instead, it is deployed on the AIST island, where it functions as an access point to the transit network domain and as a central repository to look for NSI endpoints offered by each island in FELIX.

The deployment of the module was carried out in a similar way as the other FELIX RMs, yet deployed in a single point, following a centralised fashion. TNRM is based on the eiSoil library and runs as a Flask server that exposes a XMLRPC API, which is GENIv3-compliant. Details on the deployment address, port and endpoint are provided in "Annex A", used to properly configure peers in RO and MRO.

Following on the recommendations from the Year 2 Review meeting, the support for creating inter-island connections via GRE tunnels in TNRM is under development at the time of writing this deliverable. This additional capability will increase the flexibility of the TNRM in establishing inter-domain connections using different types of technologies. This is part of the ongoing Year 3 development plans and details of the GRE-support in TNRM will be outlined in future documents.

3.6 Monitoring System

The Monitoring System (MS) is developed by KDDI and is the FELIX software module that collects the monitoring data of the resources available or provisioned in the FELIX infrastructure, to later provides it to both users and administrators. As with RO and MRO, MS can also run in two operation modes: standard MS and Master MS (MMS).

MS is responsible for the monitoring of each island and aggregate and forward it to the Master MS. Therefore, an MS instance must run per each FELIX island. As for MMS, it provides all monitoring data of FELIX infrastructure, receiving the aggregate data from the MS in each island. The MS module is deployed in PSNC, i2CAT, AIST and

KDDI islands; and the MMS module is deployed in KDDI. Also, the GUI tool (Expedient's monitoring section) is co-hosted on the MMS machine and provide the visualisation on the monitoring data to users and administrators.

In order to deploy, the source can be downloaded from the **monitoring** branch (*msjp* folder). By following the instructions available in the module and in the GitHub public wiki page, the MS or MMS module can be installed in the appropriate domain. First, dependencies are installed and a number of MySQL databases are created.

Two configuration files (*mon_api.conf* and *mon_col.conf*) are provided to be filled by the domain administrator with the Monitoring API and Monitoring Data Collector, respectively. Those files define configuration data such as the address and port of MS, connection to its MySQL DB, against the corresponding MMS (currently, MMS-JP in KDDI) or to the PerfSONAR SequelService to gather metrics of HW equipment through SNMP. Finally, API and collector services are to be run in daemon mode to allow communication to MS and to retrieve metrics from devices, respectively. As mentioned in some of the FELIX RMs (i.e., CRM, SDNRM and SERM), the information on the *JSON* configuration files for the RMs that is provided in the RO source is then used by the MS collector; therefore interrelating configuration processes of those RMs with that of MS.

3.7 Public Monitoring

Public Monitoring includes basic status information about the facility, such as whether the servers and the network connectivity are working as expected. This information is generally available within the stack in each domain (i.e. obtained through some of the exposed APIs of the different RMs).

The source for Public Monitoring is available in the GitHub public repository, under the **monitoring** branch in a folder named *public*. The configuration step consists on copying the *localsettings.py.EXAMPLE* into *localsettings.py* and filling with the appropriate access data to contact the private APIs on the required RMs. Unlike the Monitoring System, the data is not directly monitored from the resources, which is why the Public Monitoring is a standalone component not dependent on the Monitoring System and it does not need to be deployed on each FELIX island.

Since the volume of monitoring data and status information is minimal, Public Monitoring is performed and deployed in a centralised manner, from where is linked to the FELIX website to permit public access. The Public Monitoring tool runs at the iMinds facilities (`http://157.193.215.150:8080/island/`<island_name>), where it periodically polls specific APIs on the different CRM and SDNRM modules that are deployed the monitored domains. This enables retrieving topology and availability information on some of the hardware provided by each domain.

3.8 AAA

C-BAS (Certificate-based AAA for SDN Experimental Facilities) realises user access control, policy enforcement, and trust anchor for federation formation.

The source of C-BAS can be found in the **C-BAS** public GitHub repository under the *EICT* account. Instructions on configuring the module are provided in both that repository and in the FELIX GitHub wiki; and mainly correspond to the copy and modification of JSON files to define domain parameters and URNs, and to the generation of per-domain certificates.

Each island shall run its local instance of C-BAS so that it can perform a range of AAA related tasks including, but not limited to user registration, management of user credentials, management of slice and project objects and their membership, logging of user actions, etc. C-BAS is currently deployed in PSNC, EICT and i2CAT. In addition, two master islands (AIST and i2CAT) will maintain C-BAS instances running with a special configuration which allows them to serve as trusted repository of root certificates for all islands in the FELIX federation. The deployment of C-BAS in master islands is planned for Y3.

In that sense, and conforming to strict security measures, root certificates of member islands of FELIX federation will be manually configured in the master C-BAS instances which gets periodically pulled by C-BAS instances of federation member islands along with corresponding Certificate Revocation Lists (CRLs). A master C-BAS will periodically pull CRL from its member islands as well as from the other master island, e.g., master island of Europe pulls CRLs from all islands in Europe and from master island in Japan. This way, adding a new island in the federation would be as simple as configuring its root certificate in the master islands. The vice versa would hold for removing an island from the federation.

4 Testing of Functionalities per Component

In this section we present detailed information on the testing procedures performed for each FELIX software component to verify its proper behaviour and matching to design and requirements.

The aim of this chapter is not to present in depth the test results, logs or a history of the fixes carried out. Rather, we aim at describing the sequence of steps followed to verify the coherent operation of a FELIX module. We deem this information of higher interest for potential future developers of the FELIX public code base.

4.1 Resource Orchestrator

As previously described, RO is the FELIX component designed to orchestrate the end-to-end network service and to reserve and monitor the underlying resources through RMs and MS. This module has been developed from scratch in FELIX project.

The software system can operate in two *operational modes*: Resource Orchestrator (RO) per island and Master Resource Orchestrator (MRO) per continent. The only difference is the meaning of the *resources* that it can manage. Indeed, the RO is connected to the RMs in order to control the physical (i.e. Computing, SDN and Stitching) resources of the island in which it is installed. On the other hand, the MRO cooperates with the others ROs and with the TNRM in order to set-up the inter-islands paths within the continent (i.e. Europe or Japan) realising the architectural hierarchical approach described in the D2.2 document [3]. This different behaviour is simply obtained changing some parameters (e.g. *mro_enables*) of the configuration file of the module. Moreover, setting up a proper *list of peers* allows the component to activate the dedicated drivers to connect the interworking modules.

Due to its peculiarity, the (M)RO has undergone a *Sandwich Testing* procedure. That means we had firstly validated the procedures for the RO component following a *Bottom-to-Up Testing* approach, using an emulated test environment with simulated RMs. In this phase, we have verified the basic functionalities of the module, i.e. i) the compliance to the standard GENIv3 API, ii) the correct internal workflow per each method. After that, we have deployed an integration test environment composed of the RO and the *not-simulated* RMs. In this phase, we have rechecked the previous results and introduced tests for the iii) installation of the stack and iv) security aspects.

Once the previous points have been proved to work as expected, we introduced the MRO component following a *Top-to-Down Testing* approach. This allows us to verify the communication between the MRO and the controlled components (i.e. ROs and TNRM) and to validate the v) correctness of the output parameters of the northbound API, e.g. expiration dates being returned as *datetime* objects rather than as *strings*, etc.

4.1.1 Features validated

We broadly categorise those features of (M)RO that are of interest to the FELIX infrastructure as follows:

- Compliance of the (M)RO's northbound APIs with the GENIv3 standard
- Correct internal behaviour for each method exposed by the northbound API
- Correct behaviour for the resource management
- Availability of the (M)RO module and the underlying RMs

4.1.1.1 Compliance of northbound API with GENIv3

Every FELIX software module exposing the GENIv3 API **must** support the mandatory methods and arguments detailed in the GENIv3 interface [4], and **shall** support some of the optional arguments as long as they provide benefits to the experimenters or are commonly accepted by the third-party tools used by them.

In Table 4.1 we present the supported GENIv3 methods with an indication of the arguments and the return values.

Method	Ingress	Output
GetVersion	• options (not required)	• geni_api • geni_api_version • geni_credential_types • geni_ad_rspec_versions • geni_request_rspec_versions
ListResources	• credentials • options (e.g. version, available, compressed)	• advertisement RSpec
Describe	• URNs • credentials • options	• list of slivers (e.g. URN, allocation/operation status, error) • manifest RSpec
Allocate	• slice URN • credentials • request RSpec • options	• list of slivers • manifest RSpec
Renew	• URNs • credentials • expiration time • options	• list of slivers (with the new expiration time)
Provision	• URNs • credentials • options (e.g. bes-effort, end-time, users)	• list of slivers • manifest RSpec

Status	• URNS • credentials • options	• list of slivers
PerformOperationalAction	• URNS • credentials • action (e.g. "start", "stop", "restart") • options	• list of slivers
Delete	• URNS • credentials • options	• list of slivers
Shutdown	• URNS • credentials • options	• XML-RPC boolean

Table 4.1: Ingress parameters and the output values

The methods and arguments above have all been manually tested throughout several testing stages, and some of them are automatically tested in a periodic fashion through the F4F-FLS tool [5].

Besides the methods and arguments, the GENIv3 API is based on three different type of Resource Specification (*RSpec* data models): the *advertisement*, the *request* and the *manifest* RSpecs. The structure of these XML documents is described in the D3.1 deliverable [6] and schema examples can be found within the *modules/resource/orchestrator/test/delegate/geni/v3/rspecs* folder of the FELIX repository [7].

The validation of both the incoming (request) and outgoing (advertisement, manifest) RSpecs are compared and then validated against XML schemas released by the GENI group; ensuring its correct syntax.

With the two validation steps mentioned above, the (M)RO is fully compliant with the GENI system. This ensures proper federation with any other GENI-enabled testbed or tool.

4.1.1.2 Correct internal behaviour per method

In this paragraph, we summarise the internal workflow per method from an high-level point of view. It is important to note that the (M)RO has a strong relationship with the other components in terms of the exchanged messages. All these messages are printed in the log-file of the process (i.e. *modules/resource/orchestrator/log/resource-orchestrator.log*) in a pretty XML format.

In the *GetVersion* method, the (M)RO fills the XML-RPC structure with the proper values.

In the *ListResources* method, the (M)RO first verifies the user credentials and then reads from the MongoDB [8] database the information on the nodes and links for the C, SDN, SE and TN resources. At the end, the advertisement RSpec is composed and returned to the client.

In the *Describe* method, the (M)RO verifies the user credentials and retrieves the information of the slice composition from the database. Then, it sends the describe command to each involved RMs or ROs. When the responses are correctly received, it creates the manifest RSpec and the list of slivers merging the output of the corresponding message. This information is then returned to the client.

The *Renew*, *Status* and *PerformOperationalAction* methods are quite similar and we use the same template for the implementation. In these methods, the (M)RO starts verifying the user credentials and reading the slice composition from the database. After that, it sends the proper command (i.e. Renew, Status or PerformOperationalAction) and, when it receives the responses, only the list of slivers is produced and returned to the caller.

The *Allocate* method is the most complex implementation we currently have in the code. After the verification of the user credentials and the validation of the incoming request RSpec, the (M)RO tries to extend the request adding the Stitching Entity information. Here, we use a dedicated *pathFinder* module that simply lookup the database in order to fetch the missing resources description. When the request is completely reformatted, the (M)RO analyses the schema extracting the C, SDN, SE, TN sections. Each section is then introduced in the allocate command to the proper RM or RO. At the end, the (M)RO composes the manifest RSpec as sum of the received responses and fills the list of slivers structure.

The *Provision* and *Delete* methods are also used to update the Monitoring System with the information related to the slice. As usual, the user credentials are verified and the slice composition retrieved from the database. A proper command (i.e. Provision or Delete) is sent to the RMs or ROs. In case of the Provisioning method, the (M)RO forwards the slice information to the MS in order to start the metering collection for the involved resources. On the other hand, in case of the Delete method, the module informs the MS to stop the monitoring activities. The manifest RSpec and the list of slivers are formatted and returned.

The *Shutdown* command is actually not forwarded to any RM or other RO in the system, in order to prevent the power-off of any part of the FELIX testbed.

4.1.1.3 Correct behaviour for the resource management

Using a client (e.g. OMNI) that supports the GENIv3 interface, the experimenter can easily manage *resources*, i.e i) retrieve the list of available devices, servers or nodes in the testbed, ii) reserve a group of resources, iii) check the status of his/her reservation and iv) release the resources at the end of the experiments.

A correct set of GENIv3 methods should be called to realise these requirements, as briefly discussed here.

The *getVersion* and the *listResources* methods can be used to retrieve the XML document (advertisement RSpec) that describes the available resources managed by the (M)RO.

The *allocate* (with the request RSpec), the *provision* and the *performOperationalAction* (with the *start* parameter) methods can be invoked to create a reservation and to configure the resources that belong to the slice.

The *describe* and the *status* methods show the status of the provisioned resources (manifest RSpec).

The *performOperationalAction* (with the *stop* parameter) and *delete* methods allow to release the resources that became free and ready for a new reservation.

4.1.1.4 Availability of the (M)RO module and underlying RMs

The (M)RO is deployed in the servers of the FELIX testbed using some scripts that install the modules and their dependencies automatically. In the same way, other scripts are used e.g. to generate the credentials, to configure the modules, to create daemons/services, etc. Basically, the (M)RO is a python FLASK server running on a configurable port and accessible with proper credentials. That means that no particular hardware constraints are introduced.

4.1.2 Validation procedures

The validation and testing procedures carried out to test the functionalities, the availability and the connectivity against the (M)RO have been continuously implemented during both the development and deployment phases of the project. The following sections summarise the used tools and the results.

4.1.2.1 Compliance of northbound API with GENIv3

In order to verify the compliance of the (M)RO northbound interface with the GENIv3 standard, we have used the *OMNI* client.

The OMNI client is used to manually call each method validating the received output. It is worth noting that the tests were conducted having a single RM (or RO) as a remote peer of the (M)RO in order to have a simplest test environment. Moreover, the tests were repeated for every RM (i.e CRM, SDNRM, SERM and TNRM) and for the RO.

During the tests, we verified that:

- The methods must be concluded with zero errors reported on the screen
- The sliver description must have a unique identifier for the resource (URN)
- The expiration time must be reported
- The status of the resource must be as expected
- The manifest RSpec must include the details of the slice reservation

4.1.2.2 Correct internal behaviour per method

We have massively used the log file to validate the correct behaviour per each method of the interface. Using the OMNI tool, each method of the API was called and the log file deeply analysed. In case of errors, the problem was reported to the development team.

In order to facilitate this procedure, the *colorlog* module has been introduced, producing different colours for the different levels, e.g green for debug, blue for info, red for error, etc. Moreover, we have added the timestamps and a proper handler for the dumping of the messages.

As result, we have no errors in the normal test conditions.

4.1.2.3 Correct behaviour for the resource management

As for the previous case, the log file helped us to verify the correct behaviour for the resource management. We validated the exchanged messages and the fields of the XML body. Moreover, the status of the resources has been also checked to verify it was as expected.

The result is that the workflow of the operations is fully supported by the module.

4.1.2.4 Availability of MRO and RO modules and underlying RMs

The connectivity to the (M)RO is periodically evaluated, by means of a testing tool whose results are publicly available in a specific FELIX section at the Fed4FIRE's First-Level Support [5] site.

The monitoring procedure evaluates that the following conditions are appropriate:

- Connectivity of the XMLRPC server exposed by (M)RO for every domain that deploys them.
- Validity and matching of credentials (X509 certificate and key) exposed by the XMLRPC server: certificates must be valid and the same across sessions.

As said before, the tests are performed on a periodic basis. Specifically, connectivity tests are run every 5 minutes, whereas asserting the validity of certificates and retrieving the heterogeneous resources advertised by the (M)RO is performed every 10 minutes.

Two methods are called for both RO and MRO: *GetVersion* and *ListResources*. Other methods are explicitly related to the kind of the managed resource and deeply influenced by the internal logic of each RM. Therefore, the validation procedures are carried out in a per-RM fashion, rather than from the upper layers of RO and MRO.

In this aspect, we have evaluated the possibility to build a consolidated system test in order to validate the correct resource management through RO and MRO. Some aspects have been highlighted:

- the life cycle of the resources can already be tested through their respective RMs
- tests with more than one resource per time can add complexity to the testing procedures
- unclear test environment can easily generate confusion on the cause of the error

On the other hand, we admit that a more complex system test would help us to prepare a more resilient and robust orchestrator.

4.2 Software-Defined Networking Resource Manager

The Software-Defined Networking Resource Manager (SDNRM) is a FELIX software module based on a previously existing component, called OpenFlow Aggregate Manager (OFAM) and developed under the OFELIA project. As such, the core functionalities of SDNRM were already tested in that project. In FELIX, the work performed in SDNRM is related to the extension and bug fixing of its northbound GENIv3 API [4] as well as of some of the internal submodules in use and which persist the information of reserved and provisioned FlowSpace rules in the database and that communicate with the FlowVisor module.

We explain below the validation procedure and detail the tests that have been carried out to validate proper behaviour and interaction with GENI-compliant clients, whether third-party clients (OMNI, jFed) or other FELIX modules (Resource Orchestrator).

4.2.1 Features validated

We can confirm that the requirements for SDNRM, which were introduced in deliverable D2.2 [3], have been met. Specifically, the following requirements are implemented and work as expected:

- Define the special purpose controller of the testbed, which is previously configured through SDNRM's management GUI
- Approve or deny the experimenters' requests, also managed through the SDNRM's management GUI
- Authenticate experimenter credentials according to a chain-of-trust model
- Allow an experimenter to request a FlowSpace with a unique controller (unique combination of IP and port) and a subset of matching conditions to define custom rules on the selected OpenFlow-enabled switches

The northbound API of SDNRM that offers experiments the ability to request resources, as defined in the latest bullet point, is the main objective of the work carried out on SDNRM for FELIX. Specifically, the extensions, modifications and bug fixing performed were focused on the correct interoperability of the module with others, via the SDNRM's northbound API. In order to ensure proper functioning, we divided the validation and testing phases as follows:

- Compliance of SDNRM's northbound API with the GENIv3 standard
- Correct internal behaviour for each method exposed by the northbound API
- Availability of the SDNRM module and its underlying infrastructure

4.2.1.1 Compliance of northbound API with GENIv3

Having standard northbound interfaces is key for the interoperability between modules. In FELIX, we deemed convenient to use the GENIv3 API as the northbound interface of every Resource Manager (RM). The wide usage of this API makes it possible for a module exposing this interface to integrate with any existing compliant client or federation network, and requiring little development effort to that. Also, a consistent northbound API was much needed to reduce development efforts on the orchestrator module.

This being said, the exposed API must accept the required methods, the different mandatory arguments (credentials, user keys, expiration time), and options (*geni_best_effort*). The API may also allow optional ones (currently: *geni_available*, *geni_compressed*, *geni_end_time*; in the future: *geni_allocate*). The implementation, subsequent extensions and bug fixing have been performed taking the GENIv3 AM API documentation [4] as the main reference, whereas the validation has been performed manually through both OMNI and jFed.

4.2.1.2 Correct internal behaviour per method

Whilst not directly responsible for the interoperability with other modules, the internal behaviour of any Resource Manager must be sound enough to operate under a combination of options and of slivers' statuses. The validation of the proper internal behaviour has been performed by means of *unit testing*. Such tests have helped to validate the minimum component (methods and functions), internal to SDNRM.

4.2.1.3 Availability of SDNRM module and underlying infrastructure

Finally, having a monitored infrastructure helps identifying issues on the connectivity and the availability. In the case of SDNRM, the monitoring is performed at two levels: software (the SDNRM's server and implicitly, the FlowVisor component) and hardware (the OpenFlow-enabled switches). Therefore, the connectivity to the FELIX SDNRM is periodically tested through a GENIv3-enabled client; whereas the connectivity is also frequently checked against the OpenFlow-enabled switches.

4.2.2 Validation procedures

4.2.2.1 Compliance of northbound API with GENIv3

The first item on the feature list attempts to determine whether the developed API complies with the standard. The motivation of these tests is to pinpoint if there is any lack of features or options, specially those that are compulsory. The tests were carried out through an automated testing module (jFed's automated testing), already existing in Fed4FIRE, and which can be manually invoked or consulted through the First-Level Support Fed4FIRE website [5]. Whichever the case, this module performs regular calls to the northbound GENIv3 APIs of SDNRM and checks that the following items work as expected:

- Mandatory GENIv3 methods are supported by SDNRM.
- Required arguments and options for the aforementioned methods are also supported.
- Correctness and authenticity of the connections established with the SDNRM server.

The aforementioned items are tested frequently, in an automatic fashion. Specifically, the following tests are checked:

- Proper return of basic RM information.
  - Tested methods: *GetVersion*
- Datapaths and ports available for use in each island.
  - Tested methods: *ListResources*

- Proper reservation, provisioning, instantiation and description of resources.
  - Tested methods: *Allocate*, *Provision*, *PerformOperationalAction*, *Describe*
- Appropriate instantiation of the OpenFlow FlowSpace.

The validation of the GENIv3 API is performed in a very similar fashion for every FELIX RM, taking into account natural differences in the internal behaviour and thus the behaviour on a reservation or provisioning procedure. All in all, validation is pretty much the same for all of them and therefore similar sections in the rest of the document will refer to this one for further details.

In SDNRM, two additional methods are tested in a slightly different way: *Renew* (to extend the expiration date of a FlowSpace over time) and *Status* (retrieve the status for a number of FlowSpaces). The first feature was manually tested by using OMNI and jFed clients and validating that the expiration date was effectively extended (e.g. FlowSpaces would not be automatically un-granted past the original date, but only after the extended time). The second feature is internally served through the *Describe* method, which is included in the automated testing through the FLS portal and tools.

The results of these tests are available as a specific section in the FLS Fed4FIRE website. This has been commonly used by FELIX developers and infrastructure administrators to ensure proper availability of the resources in their domain.

4.2.2.2 Correct internal behaviour per method

The correctness of the workflow internal to the SDNRM submodules is also important to show consistent and robust operation. The validation process is performed here by testing incoming requests, arriving to the *manager* and going down to the *driver*. It also takes into account the returned information coming from the *manager* and going down to the *driver*. For that matter, tests have been automated to ensure a proper internal workflow. *Unit tests* for the involved methods can be found in the FELIX repository, located under the corresponding folder [9] within the *ocf* branch.

Specifically, the *unit tests* check the proper retrieval of the information on the physical resources (switches/*datapaths* and ports), exposed via the *ListResources* method. The retrieval and parsing of such resources is performed by the *Driver* component, which is located at the lowest software layer within SDNRM, therefore is directly placed above the domain's slicing controller (FlowVisor). In last instance, FlowVisor is considered by SDNRM as a sort of resource to be managed.

Besides testing the proper advertisement of the physical resources, we have implicitly validated the proper internal behaviour by requesting resources manually and through means of the OMNI and FLS tools.

4.2.2.3 Availability of SDNRM module and underlying infrastructure

Besides the proper functioning of the exposed methods and its internal processing within SDNRM, we monitor the availability of both software (connectivity to SDNRM) and hardware (switches up and running) resources.

The connectivity to the FELIX SDNRM software module is periodically tested through the FELIX section in the Fed4FIRE's First-Level Support [5] site. On the other hand, the connectivity checks against the underlying virtualisation servers is done through the Public Monitoring, which is available at `http://157.193.215.150:8080/island/<N>`, where N can be any FELIX infrastructure, namely "iMinds", "i2CAT", "PSNC", "EICT", "AIST" or "KDDI".

Availability of management software

The connectivity check performed on the SDNRM modules ensures that connections against the RM can be established.

- Connectivity of the XMLRPC server exposed by SDNRM at each island.

- Validity and matching of X509 certificates exposed by the XMLRPC server: certificates must be valid and the same across sessions.

Tests are performed on a periodic basis. Specifically, connectivity tests are run every 5 minutes, whereas the check of the validity of certs, and the diverse information of the RM and the available switches and ports is checked every 10 minutes. Finally, the basic life cycle of the resources is tested twice a day, where the reservation, provisioning, instantiation and description (*Describe* method) of granted resources are tested.

Availability of data plane

The Public Monitoring module functions as a stand-alone software tool deployed in the FELIX infrastructure. Its objective is to clearly identify the availability of the physical infrastructure supporting the FELIX SDNRM and CRM modules. In order to check connectivity, the tool communicates periodically with internal APIs offered by the SDNRM and CRM modules. Among other parameters, the interval of seconds between launches of threads is configurable, and defaults to 120 seconds.

Upon connection with a given domain, for the case of SDNRM, the Public Monitoring retrieves a detailed list of OpenFlow-enabled (physical or virtual) switches indirectly managed by SDNRM, and parses the information obtained. After interpreting the data, the internal database is updated and the website is updated with the new information, namely the name of the switch manufacturer and the specific switch model, the internal name and dpid given to the switch and, finally, the status.

4.3 Computing Resource Manager

Similarly to the SDNRM, the Computing Resource Manager (CRM) is a FELIX software module based on a previously existing component, called Virtualisation Technology Aggregate Manager (VTAM) and developed under the OFELIA project. As such, its core functionalities were already tested there. The work in SDNRM performed in FELIX is related to the extension and the bug fixing of its northbound GENIv3 API [4] and of some of the internal submodules being used to attend the requests coming from the northbound interface.

In the following sections we detail the validation and testing procedures carried out to ensure proper interworking with GENI-compliant clients, whether third-party clients (OMNI, jFed) or other FELIX modules (Resource Orchestrator).

4.3.1 Features validated

The main extensions and improvements performed on CRM deal with the proper interoperability of this module and others via the northbound API of the CRM. To validate its correct functioning, we divided the validation and testing phases as follows:

- Compliance of CRM's northbound API with the GENIv3 standard
- Correct internal behaviour for each method exposed by the northbound API
- Availability of the CRM module and its underlying infrastructure

4.3.1.1 Compliance of northbound API with GENIv3

As explained previously for the SDNRM validation, exposing standard northbound interfaces is key for the interoperability between modules. RMs in FELIX use the GENIv3 API for that matter, which ensures easy integration with other infrastructures and almost direct usage by any client supporting GENIv3 API.

The API exposed by CRM must accept the same required methods and arguments and shall accept some of the optional ones; as described in SDNRM's *Compliance of northbound API with GENIv3* section. The validation of the interface has been carried out in the same way as well.

4.3.1.2 Correct internal behaviour per method

In the same way as explained in SDNRM section, the internal behaviour of the CRM is verified by means of *unit tests* over its methods and functions to assure it can operate under different combinations of options and slivers' statuses.

4.3.1.3 Availability of CRM module and underlying infrastructure

The availability of software and physical resources is another item to have in mind when it comes to testing. It is necessary to identify possible issues with the connectivity to the software module or its underlying hardware, as well as it is important to detect malfunctioning hardware. Therefore, the connectivity to the FELIX CRM is periodically tested through a GENIv3-enabled client; whereas the connectivity is also frequently checked against the virtualisation servers.

4.3.2 Validation procedures

4.3.2.1 Compliance of northbound API with GENIv3

The implementation of this interface and its subsequent extensions and bug fixing was based on the GENIv3 AM API [4] documentation. On the other hand, the validation process must test the support for, at least, the mandatory methods, arguments and options. During validation, we checked that the following items work as expected:

- Mandatory GENIv3 methods are supported by CRM.
- Required arguments and options for the aforementioned methods are also supported.
- Correctness and authenticity of the connections established with the CRM server.

The aforementioned items are tested frequently, in an automatic fashion. Specifically, the following tests are checked:

- Proper return of basic RM information (*GetVersion* method) and virtualisation servers available for use in each island (*Listresources* method).
  - Tested methods: *GetVersion*, *ListResources*.
- Proper reservation, provisioning, instantiation and description of resources.
  - Tested methods: *Allocate*, *Provision*, *PerformOperationalAction*, *Describe*.
- Appropriate instantiation and key contextualisation into the VMs.

However, this does not cover the full set of methods of the API. Two methods are missing: *Renew* (to extend the expiration date of a VM over time) and *Status* (retrieve the status of one or multiple VMs). The first feature has been manually tested by using OMNI and jFed clients and validating that the expiration date had been effectively extended (e.g. VMs would not be automatically deleted passed the original date, but only after the extended time). The second feature is internally served through the *Describe* method, which is already covered by automated testing through the FLS portal and tools.

The results of these tests are available as a specific section in the FLS Fed4FIRE website [5]. Such site is commonly used by FELIX developers and infrastructure administrators to ensure proper availability of the resources in their domain.

4.3.2.2 Correct internal behaviour per method

The second item seeks to verify the correctness of the internal workflow of requests. That is done by testing incoming requests that arrive to the *manager* and traverse all way down until arriving at the *driver*. It also takes into account the returned information coming from the *manager* and going down to the *driver*. For that matter, tests have been automated to ensure a proper internal workflow. *Unit tests* for the involved methods can be found in the FELIX repository, located under the corresponding folder [10] within the *ocf* branch.

Looking into detail, the *unit tests* in CRM look for a proper behaviour/output on the following:

- Management of unique identifiers (URN, HRN) for resources: ensure correct retrieval of the identifiers, appropriate parsing of authority, slice and sliver entities, and correct translation of URNs into HRNs and vice-versa.
- Management of reservations: validate that reservations are assigned the default expiration time (1 hour) and ensure proper type handling during the management of the projects and slices.
- Advertising resources: return (un)available servers, return slivers per slice, etc.
- Management of slivers: correctly starting, stopping, rebooting, deleting and renewing resources.
- Internal management: ensure credentials are verified correctly, handlers are set up as expected, and resource specifications are accurately translated into an internal representation (model) used within internal components.

The previous tests cover pretty much the full experimenter life cycle, yet further manual tests have been performed during the integration phase with the Resource Orchestrator (RO). These manual tests have been carried out both manually (using a third-party client, such as jFed or OMNI) and also by using the RO to proxy requests to its managed RMs.

4.3.2.3 Availability of CRM module and underlying infrastructure

Besides the proper functioning of the exposed methods and its internal processing within CRM, we monitor the availability of both software (connectivity to CRM) and hardware (virtualisation server) resources. The connectivity to the FELIX CRM is periodically tested through the FELIX section in the Fed4FIRE's First-Level Support [5] site. The connectivity checks against the required, underlying virtualisation servers is done through the Public Monitoring, which is available at `http://157.193.215.150:8080/island/`<N>, where N can be any FELIX infrastructure, namely "iMinds", "i2CAT", "PSNC", "EICT", "AIST" or "KDDI".

Availability of management software

The monitoring of the software part validates the following:

- Connectivity of the XMLRPC server exposed by CRM at each island.
- Validity and matching of X509 certificates exposed by the XMLRPC server: certificates must be valid and the same across sessions.

Tests are performed on a periodic basis. Specifically, connectivity tests are run every 5 minutes, whereas the check of the validity of certs, and the diverse information of the RM and the available servers is checked every 10 minutes. Finally, the basic life cycle of the Virtual Machines (VMs) is tested twice a day. To that end, the tests issue reservation, provisioning, instantiation and description (*Describe* method) commands over the CRM. In the case of CRM, and implicit to the instantiation process, key contextualisation is performed on the VMs to grant access to a number of required users.

Availability of data plane
The Public Monitoring module was already introduced in the section for SDNRM. As mentioned there, that tool polls periodically every island for information on physical resources managed by both CRM and SDNRM modules. As for the case of CRM, once the Public Monitoring tool connects to a specific internal API, the Public Monitoring retrieves a detailed list of virtualisation servers, indirectly managed and contacted by CRM, and parses the information obtained. After interpreting the data, the internal database is updated and the website is updated with the set of information, namely the model and internal name given to the server, its RAM, the OS running in the server (defaults to Debian Squeeze), the virtualisation technique (defaults to XEN), the OS running in the server and, finally, the availability (up/down) of the server.

4.4 Stitching Entity Resource Manager

The Stitching Entity Resource Manager (SERM) is a component, developed from scratch in the context of the FELIX project, that participates in the inter island network provisioning. While the Transit Network Resource Manager (TNRM) activates the inter-island links the SERM is in charge of interconnecting the links (dynamically created by TNRM or statically by other network tools) with the local SDN island. In fact, this is ensured by the switching rules configuration (e.g. adding OpenFlow flows) on the switching device (Stitching Entity). The stitching concept has been introduced by the GENI (Global Environment for Network Innovations) initiative [11], the draft description can be found in the "GENI Network Stitching -- Overview" document [12].

The SERM, like other FELIX modules, offers a similar northbound GENIv3 API. The SERM allows an experimenter to request, update and delete stitching resources (in fact the switching rules). This module also acts as an interface between the RO (Resource Orchestrator) and Stitching Entity (switching device), allowing to manage the latter.

4.4.1 Features validated

The features of SERM that are in scope of validation are:

- Compliance of the SERM's northbound APIs with the GENIv3 standard
- Correct internal behaviour for each method exposed by the northbound API
- Correct behaviour for the resource management
- Availability of the SERM module

4.4.1.1 Compliance of northbound API with GENIv3
As for all the Resource Managers in the FELIX software stack, the SERM **must** support the mandatory methods and arguments from the GENIv3 interface, as detailed in [4]. The list of GENIv3 methods and arguments that are supported by every RO and RM is detailed in Table 4.1.

The GENIv3 API is based on three different type of Resource Specification: the *advertisement*, the *request* and the *manifest* RSpecs. The detailed structure of the XML documents are presented in the D3.3 deliverable. The example schemas can be found in the source code at
modules/resource/manager/stitching-entity/test/delegate/geni/v3/rspecs folder of the FELIX repository.

Each incoming RSpec is compared and then validated against XML schemas released by the GENI group. That means that the SERM is fully compliant with the GENI system allowing us to provide a federation with the GENI testbed.

4.4.1.2 Correct internal behaviour per method

In the same way as in the (M)RO section, the expected effect per method for the SERM is presented. It is important to note that both components have a strong relationship and the behaviour of each method is very similiar. Nevertheles, a description of each method behaviour is also presented in order to provide the specific details. Furthermore, all the messages are printed in the log-file of the process (i.e. modules/manager/stitching-entity/log/stitching-entity.log) in a pretty XML format.

The *GetVersion* method fills the XML-RPC structure with the proper values.

In the *ListResources* method, the SERM first verifies the user credentials and then reads from the mongoDB database the information on the nodes and links for the SE resources. At the end, the advertisement RSpec is composed and returned to the client.

In the *Describe* method, the SERM verifies the user credentials and retrieves the information of the slice composition from the database. Then, it creates the manifest RSpec and the list of slivers merging the output of the corresponding message. This information are then returned to the client.

The *Renew*, *Status* and *PerformOperationalAction* methods have the same behaviour as in the (M)RO.

In the *Allocate* method first the verification of the user credentials and the validation of the incoming request RSpec is being processed. Then SERM checks in its local mongoDB database if the requested resources (VLANs per ports) are available to reservation and/or if are not currently reserved by the other reservation. If the resources are free then SERM allocates them in the database and prepares the manifest RSpec as a list of slivers structure.

The *Provision* method is very specific for each component. In SERM, first the verification of the user credentials and the validation of the incoming request RSpec is being processed. Then the method changes the resource state in the local mongoDB database in order to allow perform operational actions on them (in fact to allow on inserting new VLAN translations on the switching devices). The manifest RSpec and the list of slivers are formatted and returned.

Finally, in *Delete* method first the verification of the user credentials and the validation of the incoming request RSpec is being processed. Then the VLAN translations rules are being removed from the switching device and the resources are being released in the local mongoDB database.

4.4.1.3 Availability of SERM

The SERM is deployed in the servers of the FELIX testbed using some scripts that install the modules and their dependencies automatically. In the same way, other scripts are used e.g. to generate the credentials, to configure the modules, to create daemons/services, etc. Basically, the SERM is a python Flask server running on a configurable port and accessible with proper credentials. That means that no particular hardware constraints are introduced.

4.4.2 Validation procedures

The validation and testing procedures carried out to test the functionalities, the availability and the connectivity against the SERM have been continuously implemented during both the development and deployment phases of the project. The following sections summarise the used tools and the results.

4.4.2.1 Compliance of northbound API with GENIv3

In order to verify the compliance of the SERM northbound interface with the GENIv3 standard the *OMNI* client has been used as a tool for invoking the tested methods.

The OMNI client is used to manually call each method validating the received output. It is worth noting that the tests were conducted having a single SERM in order to have a simplest test environment. Moreover, the tests were repeated with invoking the methods by the RO

The tests verified that:

- the methods concluded with zero errors reported on the screen

- the sliver description had a unique identifier for the resource (URN)
- the expiration time was reported
- the status of the resource was as expected
- the manifest RSpec included the details of the slice reservation

4.4.2.2 Correct internal behaviour per method

During the tests the log file had been used to validate the correct behaviour per each method of the interface. Using the OMNI tool, each method of the API was called and the log file deeply analysed. In case of errors, the problem was fixed by the SERM developers.

The correctness of the VLAN translation rules installation on the switching device had been confirmed by using the Ryu REST API and the method that displays all the OpenFlow rules. In first phase of the tests the software Open VSwitch has been used as a Stitching Entity. In the next phase the proper VLAN translations on the Juniper MX80 with the OpenFlow support was also tested.

As result, no errors in the normal test conditions occurred.

4.4.2.3 Availability of the SERM module

The connectivity to the SERM is periodically evaluated, by means of a testing tool whose results are publicly available in a specific FELIX section at the Fed4FIRE's First-Level Support [5] site.

The monitoring procedure evaluates that the following conditions are appropriate:

- Connectivity of the XMLRPC server exposed by SERM for every domain that deploys them.
- Validity and matching of credentials (X509 certificate and key) exposed by the XMLRPC server: certificates must be valid and the same across sessions.

As it was said before, the tests are performed on a periodic basis. Specifically, connectivity tests are run every 5 minutes, whereas asserting the validity of certificates and retrieving the heterogeneous resources advertised by the SERM is performed every 10 minutes.

Two methods are called for SERM: *GetVersion* and *ListResources*.

4.5 Transit Network Resource Manager

The Transit Network Resource Manager (TNRM) is a new module developed from scratch within FELIX by AIST. This module is based on the *eiSoil* library [13], presented in [14]. *eiSoil* runs as a Flask server that exposes a GENIv3-compliant XML-RPC API. TNRM can support various transit network services such as NSI Connection Service v2 (NSI) [15] and GRE tunnelling. The first implementation supports NSI using the GridARS library, which is one of the reference implementations of NSI.

The TNRM implementation translates an incoming GENIv3 request into the related NSI request, and sends it to an NSI aggregator that coordinates the connection of inter-domain NSI-based transit networks. After the TNRM receives the request result from the NSI aggregator, the TNRM returns the NSI result as a GENIv3 response to the requester. In the following sections we describe the validation and testing procedures carried out to ensure proper interworking with GENI-compliant clients, whether third-party clients (OMNI, jFed) or other FELIX modules (Resource Orchestrator), and an NSI aggregator.

4.5.1 Features validated

The northbound API of the TNRM, which offers experimenters the ability to request resources, makes it the main objective of the work carried out within the RMs for FELIX.

The development of the TNRM module focused on its correct interoperability with other components, via the TNRM's northbound API and the target transit network service API for underlying infrastructure. As in the case of other RMs, we divided the validation and testing phases as follows in order to ensure proper functioning:

- Compliance of TNRM northbound API with the GENIv3 standard
- Correct internal behaviour for each method exposed by the northbound API
- Availability of the TNRM module and its underlying infrastructure

4.5.1.1 Compliance of northbound API with GENIv3

The TNRM also utilises the GENIv3 API as northbound API to facilitate the interoperation between FELIX modules. This exposed API must accept the required methods, the mandatory arguments (credentials, user keys, expiration time), and options (geni_best_effort).

The API exposed by TNRM must accept the same required methods and arguments and shall accept some of the optional ones; as described in SDNRM's *Compliance of northbound API with GENIv3* section. The validation of the interface has been carried out in the same way as well, being the only difference in validation the use of a centralised instance for TNRM, within AIST premises.

4.5.1.2 Correct internal behaviour per method

As for the case of other RMs, the TNRM is not directly responsible for the interoperability with other modules. However, the internal behaviour of any Resource Manager must be sound enough to operate under a combination of options and of slivers' statuses.

Unit testing was used to validate the minimum methods and functions internal to the TNRM, which correspond with the mandatory methods and arguments of any FELIX RM.

4.5.1.3 Availability of TNRM module and underlying infrastructure

A monitored infrastructure helps identifying issues on the connectivity and the availability. However, in the case of the TNRM the monitoring of the NSI control plane is performed at the NSI aggregators. Therefore, the connectivity to the FELIX TNRM is periodically tested through a GENIv3-enabled client in the FELIX section of F4F-FLS.

Note also that the NSI-based Transit Network is provided by multiple third-party domains external to FELIX and it is not possible to confirm the end-to-end data plane connectivity automatically at this point.

4.5.2 Validation procedures

4.5.2.1 Compliance of northbound API with GENIv3

In order to verify the compliance of the TNRM northbound interface with the GENIv3 standard, we used the OMNI client to manually call each method and validate the received output. We checked that the following items work as expected:

- Mandatory GENIv3 methods are supported by the TNRM.
- Required arguments and options for the aforementioned methods are also supported.
- Correctness of the NSI connections established with the NSI aggregator.

Specifically, the following tests are checked:

- Proper return of basic RM information
  - Tested methods: *GetVersion*
- STPs (Service Termination Points) available for inter-island connections over NSI networks.
  - Tested methods: *ListResources*
- Proper reservation, provisioning, instantiation, description, status, extension of the expiration date and delete of resources.
  - Tested methods: *Allocate*, *Provision*, *PerformOperationalAction start/stop*, *Describe*, *Status*, *Renew*, *Delete*.
- Appropriate instantiation of the requested NSI connection.

4.5.2.2 Correct internal behaviour per method

Again, similar to the other RMs, the correctness of the workflow internal to the TNRM submodules is also important to show consistent and robust operation. The validation process performed here tested incoming requests, arriving at the manager and passed to the NSI aggregator. Any returned information from the manager and passed to the NSI aggregator is also accounted for. It is important to note that these methods may fail because the underlying NSI infrastructure is provided by multiple domains external to FELIX that are not necessarily dedicated for the FELIX testbed. This does not indicate that the internal behaviour of the TNRM is at fault, but rather also serves to demonstrate that the TNRM can correctly pass the information to the northbound side. Because of the multi-domain path of NSI networks, these tests have been conducted manually to ensure a proper internal workflow using the OMNI tool.

4.5.2.3 Availability of TNRM module and underlying infrastructure

In addition to the proper functioning of the exposed methods and the correct internal behaviour within the TNRM, we monitor the availability of the connectivity to the TNRM through periodical tests in the FELIX section of Fed4FIRE's First-Level Support [5] site.

The monitoring procedure evaluates that the following conditions are satisfied and a connection can be made:

- Connectivity of the XMLRPC server exposed by TNRM at the AIST island.
- Validity and matching of X509 certificates exposed by the XMLRPC server: certificates must be valid and the same across sessions.

Tests are performed on a periodic basis, similar to other RMs. Connectivity tests are run every 5 minutes. Checking the validity of certificates, the diverse information of the RM, and the available STPs is performed every 10 minutes. The two methods that are called for the TNRM for testing are: *GetVersion* and *ListResources*. Again, because the NSI-based transit network is provided by multiple third-party domains external to the FELIX infrastructure, there is no common monitoring system to check for its data plane. As a result, the TNRM in FELIX does not support data plane monitoring at this time.

4.6 Monitoring System

Within FELIX we have identified two types of monitoring: the Monitoring System (or Infrastructure Monitoring, as named in D2.2 [3]) and the Public Monitoring (or Facility Monitoring in D2.2).

The **Monitoring System** deals with the monitoring of the actual resources, either available (physical) or provisioned (within a slice). The **Public Monitoring**, on the other hand, includes basic status information about the facility, such as whether the virtualisation servers and network devices are up or not, and also whether there is proper connectivity to testbeds and their FELIX software modules.

The required features for the Infrastructure Monitoring have been developed within the **Monitoring System** (MS). In a similar fashion to the Resource Orchestrator, the Monitoring System can work either as a Master MS, in the upper layer; or as normal, single-domain MS (intermediate layer).

The Monitoring System collects the different types of information defined in D2.2 [3] for the Infrastructure Monitoring, that is: i) the resources available and advertised by each facility (that is, physical resources and topology), and ii) a subset of such available resources, which are provisioned to the experimenter (that is, slice resources). The Monitoring System also collects metrics information from these physical resources (computing nodes and network devices) that have been granted to a specific experimenter.

In order to obtain the information of the physical topology, the MS and MMS contact RO and MRO, respectively. Given that RO is the entry point for resource request to any island or domain, the RO maintains an overview on the managed resources. Any other physical information that is not obtained through the managed RMs is retrieved from the configuration files, available per domain and which must be previously filled by the domain's administrator. With all this information available, RO retrieves the physical information, then parses it to comply with the specific schema expected by MS. Finally, it posts such information to a specific REST API endpoint exposed by the MS. Similarly, MRO obtains such data from RO and aggregates the information, which will be posted to the MMS.

Finally, to obtain the resources provisioned by a user in a given slice, the RO is again intercepting the experimenter's request and pushing a properly formatted XML document with the requested topology and resources, with details such as management information for each resource (so the MS is able to access them for retrieving metrics), name of the slice and the user, time-stamp with the last time this information was retrieved, etc. The MS shall evaluate the correctness of the received information and store it on its internal database. When a user accesses the Expedient GUI, the user will be able to see both the requested resources for a given slice, as well as every resources available.

In the following sections we will explain which design requirements or features have been validated, and how.

4.6.1 Features validated

We broadly categorise those features of MS and MMS that are of interest to the FELIX infrastructure as follows:

- Correct behaviour of the MS API defined in FELIX project
- Correct internal behaviour for each endpoint exposed by the MS API
- Availability of the MMS and MS modules and the underlying components

4.6.1.1 Correct behaviour of the MS API defined in FELIX project

We defined the MS API so as to interact with other entities (GUI, MMS, MS and RO) for the monitoring of FELIX infrastructure. In Table 4.2 we present the endpoint with an indication of the ingress information and the output values.

Endpoint	Method	Contents	Returns
http://<island_IP>:8448/topology/	GET		List of all registered topology
http://<island_IP>:8448/topology/	POST	Topology	
http://<island_IP>:8448/topology/physical/	GET		Physical Topology
http://<island_IP>:8448/topology/slice/<sliceID>	GET		Slice Topology

http://<island_IP>:8448/monitoring/<RM>	POST	Monitoring Data	
http://<island_IP>:8448/monitoring/<RM>/physical/	GET		Monitoring Data of physical resources
http://<island_IP>:8448/monitoring/<RM>/slice/	GET		Monitoring Data of resources provisioned for the slice

Table 4.2: MS API endpoints with an indication of the arguments accepted and the return values

The API handles the two types of data accepted through the API:

- *Topology* describes the topologies of the physical FELIX infrastructure and also of the slice provisioned by the users.
- *Monitoring* is the actual monitoring data (metrics) of resources. We already described the structure of such XML documents in [16]. The example data document are also provided.

The incoming and outgoing XML schemas are properly validated against the agreed schemas, defined for interoperation with MS and MMS. Information errors on parsing can be found in various log files, filled up by the Monitoring Service.

4.6.1.2 Correct internal behaviour per endpoint

In this section, we summarise the internal workflow associated with each endpoint, from a high-level point of view.

- */monitoring-system/topology* endpoint: (M)MS generates XML structures with all registered topology information (both for physical infrastructure and slice), when the endpoint is accessed through HTTP's GET method. When some topology information is provided to the endpoint (HTTP's POST method), (M)MS first verifies the XML schema and parses it to extract the expected data; then save the topology information to the SQL database. In the event of invalid or incomplete data, errors are both written in the log and an error message is provided as result.
- */monitoring-system/monitoring/<RM>* endpoint: (M)MS first verifies the XML schema and parses it, then save the time-series monitoring data of each Resource Manager (i.e. RM, SDNRM, SERM and TNRM) to the SQL database. When the endpoint is accessed by an HTTP GET method (specifically, */monitoring-system/monitoring/<RM>/physical*) and sliceID, (M)MS search the monitoring data of the FELIX infrastructure resources related to the specified slice from the database and returns the XML structure with the data to the client, if available. When the */monitoring-system/monitoring/<RM>/slice* endpoint is accessed, (M)MS looks up the database for the monitoring data of the allocated resources (for the specified slice), to finally return the XML-formatted information with the metrics to the client.

4.6.1.3 Correct internal behaviour with other components

MS collects the monitoring data by interacting with third-party monitoring tools that retrieve measurements directly from the physical hardware, server and network equipment. We use the following tools at this point:

- perfSONAR: widely deployed in the Research and Education Network as measurement tool, is used for the monitoring of network equipment.

- Zabbix: common monitoring tools for server, is used for the servers.
- NSI monitoring: proprietary monitoring tools for NSI, is used for the stats monitoring of the NSI connection.

4.6.1.4 Availability of MMS and MS modules and underlying components

The availability of the MS API can be checked by other monitoring tools (such as the visualisation tool, extended for monitoring purposes), yet the focus of the availability checks is prominently focused on the modules that manage resource and request management.

4.6.2 Validation procedures

The validation and testing procedures carried out to test the functionalities, the availability and the connectivity against the (M)MS have been continuously implemented during both the development and deployment phases of the project. The following sections summarise the used tools and the results.

4.6.2.1 Correct behaviour of the MS API defined in FELIX project

In order to verify the behaviour of the MS API, we have used the CURL client (commonly used as a client tool for REST APIs), so as to manually call each API and validate the received output. Specifically, we checked that the following items work as expected:

- The methods in the API must conclude its operation without any errors
- All topology and monitoring data posted through the API must be properly evaluated and saved correctly into the database
- The proper data must be returned in response to the request from client
- The returned data must be correct according to the defined XML schema

4.6.2.2 Correct internal behaviour per endpoint

We have massively used the log file to validate the correct behaviour per each endpoint of the API. Using the CURL client, each API was called and the log file was deeply analysed. In case of errors, the problem was reported to the development team.

In order to investigate in more detail, the MS outputs more detail log when it runs debug mode. As a result, we have no errors in the normal testing conditions.

4.6.2.3 Correct internal behaviour with other components

The monitoring data retrieved from third-party monitoring tools is saved to the SQL database through the MS API. We have checked the consistency using data of third-party monitoring tools and verified that correct data is stored in the database. In addition, the developed visualisation tool can be used to check the availability of the monitoring data.

4.6.2.4 Availability of MS and MMS modules and underlying components

We can monitor the availability of MS by checking the HTTP service, since API is served via web. Some monitoring tools, such as Zabbix, can be used for that as well; but we do not use those at this moment.

4.7 Public Monitoring

Inside the **Public Monitoring** (or Facility Monitoring), we divided the tasks it performs in two, according to their scope:

- Testing availability of software modules
- Testing connectivity of hardware devices

The first one is covered by the First-Level Support site used at Fed4FIRE [5]. This has been possible after due integration with the FELIX SW modules. Finally, the second check is performed within the Public Monitoring tool, which provides status and other detailed information in the form of a separate, public website.

4.7.1 Features validated

We broadly categorise those features of Public Monitoring that are of interest to the FELIX infrastructure as follows:

- Correct behaviour of monitoring software modules.
- Correct behaviour of monitoring hardware connectivity.
- Availability of the Public Monitoring modules.

4.7.1.1 Correct behaviour of monitoring software modules

In this paragraph, we summarise the internal workflow monitoring software modules from a high-level point of view.

The desired software component to be monitored is registered within the database, storing the necessary parameters to generate a correct call to the software component. Those components are a subset of FELIX RMs and are described in the sections above.

The main page shows an overview of the general status of the different software components deployed on each island with the information of the last running tests. Each test call is executed every 5 min and the information is stored and kept as a historic archive on the database to keep track of the availability.

4.7.1.2 Correct behaviour of monitoring hardware connectivity

We defined an endpoint for a web service that returns a formatted HTML webpage with the general overview of the island connectivity. In Table 4.3, the URI format is presented:

Endoint	Method	Arguments	Returns
`http://157.193.215.150:8080/island/<name>`	GET	name = {"iMinds", "i2CAT", "PSNC", "EICT", "AIST", "KDDI"}	List of resources within a given island

Table 4.3: Public Monitoring web service URI

4.7.1.3 Availability of Public Monitoring modules

The Public Monitoring tool is software a standalone component. As such, it can be installed on any physical or virtual machine without introducing any particular hardware constraints. To access the interface of the Public Monitoring, an HTML standard browser can be used. In case of unavailability, an error message is displayed to the user.

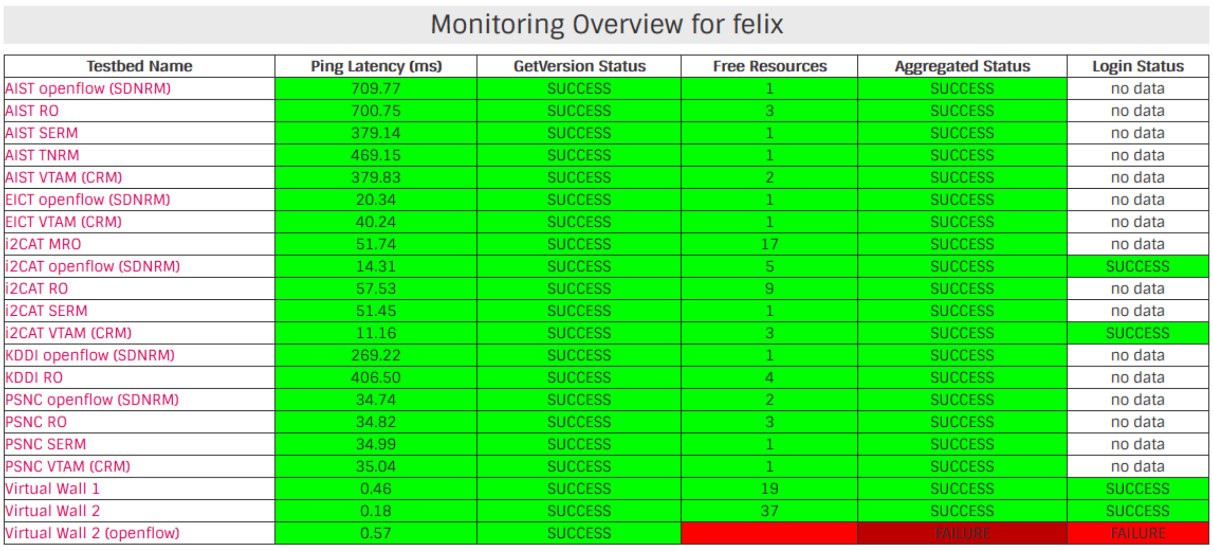
Also, it is possible to use third-party web monitoring tools to monitor the availability of the service.

4.7.2 Validation procedures

This section explains the validation procedures carried out to test the correct behaviour of the Public Monitoring.

4.7.2.1 Correct behaviour monitoring software modules

The Public Monitoring for software modules is an integration from Fed4FIRE where *unit tests* are used to validate the behaviour of the core functions. Specific configurations inside FELIX project are verified using manual testing to ensure test requests have the correct parameters. Figure 4.1 shows the main page of the Public Monitoring with the status of different tests performed for each module.



Monitoring Overview for felix

Testbed Name	Ping Latency (ms)	GetVersion Status	Free Resources	Aggregated Status	Login Status
AIST openflow (SDNRM)	709.77	SUCCESS	1	SUCCESS	no data
AIST RO	700.75	SUCCESS	3	SUCCESS	no data
AIST SERM	379.14	SUCCESS	1	SUCCESS	no data
AIST TNRM	469.15	SUCCESS	1	SUCCESS	no data
AIST VTAM (CRM)	379.83	SUCCESS	2	SUCCESS	no data
EICT openflow (SDNRM)	20.34	SUCCESS	1	SUCCESS	no data
EICT VTAM (CRM)	40.24	SUCCESS	1	SUCCESS	no data
i2CAT MRO	51.74	SUCCESS	17	SUCCESS	no data
i2CAT openflow (SDNRM)	14.31	SUCCESS	5	SUCCESS	SUCCESS
i2CAT RO	57.53	SUCCESS	9	SUCCESS	no data
i2CAT SERM	51.45	SUCCESS	1	SUCCESS	no data
i2CAT VTAM (CRM)	11.16	SUCCESS	3	SUCCESS	SUCCESS
KDDI openflow (SDNRM)	269.22	SUCCESS	1	SUCCESS	no data
KDDI RO	406.50	SUCCESS	4	SUCCESS	no data
PSNC openflow (SDNRM)	34.74	SUCCESS	2	SUCCESS	no data
PSNC RO	34.82	SUCCESS	3	SUCCESS	no data
PSNC SERM	34.99	SUCCESS	1	SUCCESS	no data
PSNC VTAM (CRM)	35.04	SUCCESS	1	SUCCESS	no data
Virtual Wall 1	0.46	SUCCESS	19	SUCCESS	SUCCESS
Virtual Wall 2	0.18	SUCCESS	37	SUCCESS	SUCCESS
Virtual Wall 2 (openflow)	0.57	SUCCESS		FAILURE	FAILURE

Figure 4.1: Public Monitoring

4.7.2.2 Correct behaviour monitoring hardware connectivity

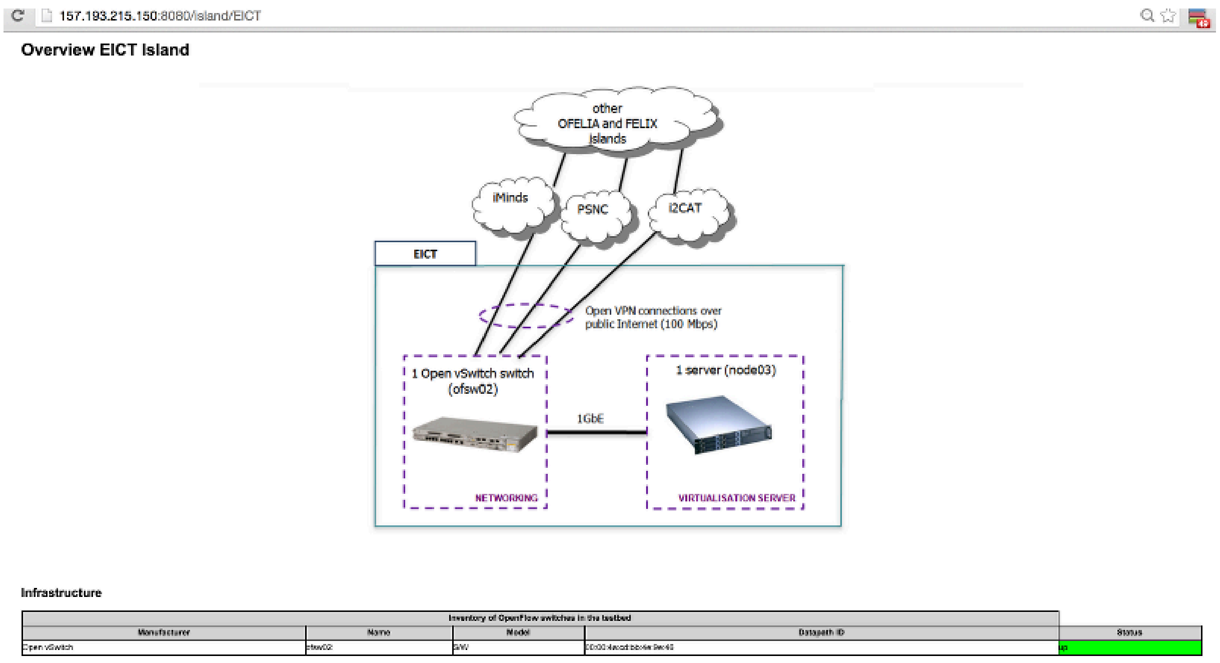
Manual testing is mainly used to check the correct performance of the monitoring connectivity. Activity is registered using log files, in case problems or bugs appear is possible to trace the problem. Figure 4.2 shows an overview of a FELIX island with a schema of the deployed components and the status of its infrastructure.

4.7.2.3 Availability of Public Monitoring modules

It is possible to monitor the availability of Public Monitoring by checking the HTTP, as it is provided through a web server. Some monitoring tool such as Zabbix or other third-party web monitoring tool can be used for that.

4.8 AAA

The AAA services in FELIX are provided by the C-BAS module, which implements an extended version of Common Federation API [17]. The user access to FELIX infrastructure is possible through three popular user-agents namely OMNI [18], jFed [19] or Expedient [20]. The following subsections explain how the implemented features of C-BAS have been validated.


157.193.215.150:8080/island/EICT
other OFELIA and FELIX islands
iMinds
PSNC
i2CAT
EICT
Open VPN connections over public Internet (100 Mbps)
1 Open vSwitch switch (ofsw02)
1GbE
1 server (node03)
NETWORKING
VIRTUALISATION SERVER

Overview EICT Island

Infrastructure

Inventory of OpenFlow switches in the testbed				
Manufacturer	Name	Model	Datapath ID	Status
Open vSwitch	ofsw02	S/W	00:00:4e:cd:bb:4e:9e:46	up 🟩

Figure 4.2: Hardware monitoring

4.8.1 Features validated

- Compliance with Common Federation API
- Correct internal behaviour per method
- Compatibility with OMNI
- Compatibility with jFed
- Integration with Expedient
- Availability of Clearinghouse module

4.8.1.1 Compliance with Common Federation API

C-BAS is fully compatible with the Common Federation API, purposed by GENI [21] and FIRE [22]. This makes C-BAS a pluggable clearinghouse that can be accessed through popular user-agents, like OMNI, jFed and so on. Moreover, compliance with the Common Federation API simplifies the task of federating FELIX islands with other GENI/FIRE islands.

Table 4.4 gives an overview of few methods from Common Federation APIs. In general there are four methods (*create*, *update*, *lookup* and *delete*), which can be executed on five types of objects (*slice*, *sliver info*, *project*, *member* and *key*) with the appropriate arguments.

In addition, C-BAS also supports slice/project member services through API methods, which are listed in Table 4.5. Arguments within parenthesis are optional.

Method	Method Arguments					Method Output
	Slice	Sliver Info	Project	Member	Key	
Create	Name (Description) (Expiry date)	Sliver URN, Slice URN, Creator's URN, Aggregate's URN, Expiry date, (Creation date)	Name Expiry date (Description)	Public information, (Public SSH key), (Privileges)	Member URN, Type, Public key, (Description)	Dictionary of fields associated with the creation of object
Update	Description or Expiry date	Expiry date	Description or Expiry date	Public information, Privileges, or membership expiry date	Description	None
Lookup	Slice URN, Slice UID, Expired, or Project URN	Sliver URN, Slice URN, Creator's URN, Aggregate's URN	Project URN, Project UID, or Expired	Member URN, Member UID, Username, Email, First name, or Last name	Member URN, Key ID, Type, Public key, or Description	List of dictionaries indexed by object URN with field/value pairs for each matching object
Delete	-Not allowed-	Sliver URN	Project URN	-Not allowed-	Key ID	None

Table 4.4: Methods of Common Federation API version 2

Method	Description	Arguments	Output
modify_members	Add, remove members or change their role with respect to given object	Object URN, Member URN, Member role	None
lookup_members	Search members matching given criteria	Object URN	List of dictionaries of member URN/role pairs
lookup_for_members	Search objects to which a given member belongs	Member URN	List of dictionaries of object URN/role pairs

Table 4.5: Methods for slice/project member service

4.8.1.2 Correct internal behaviour per method

Before executing any requested operation, C-BAS must perform a number of checks to validate the request. This includes authentication and authorisation, sanity checks on passed argument values as well as scrutiny of arguments to circumvent malicious actions. If any of the aforementioned checks fails, the method execution is aborted and an error message is returned to the caller. Here follows a list of methods, along with the measures taken before their execution.

- **create**: When creating an object, the identity of the caller and its privileges must be verified. Passed arguments must be checked against the allowed and required arguments for the create method. After this, a lookup must be performed to ensure that the requested object does not already exist in the system. When creating a slice, reuse of a name is allowed only if the previous slice with the same name has expired.
- **update**: To authorise an update method call, the caller must be a member of the object with sufficient privileges (e.g., member of a project) or the object must belong to the caller (e.g., SSH key of a member). Only certain fields of an object can be updated after its creation, e.g., username of a member cannot be modified. If the passed arguments are allowed to modify, a lookup must be performed to check if the requested object exists, after which the actual update can be performed.
- **lookup**: Lookup call should not require any authorisation. This is because only public information about the objects and members is stored at C-BAS. However, care must be taken that internal information of C-BAS (like database IDs) must not reveal to the caller.
- **delete**: This method call must be supported only for sliver info, project and key objects. Slice objects must not be deleted because there is no authoritative way to know if all slivers belonging to that slice have been released. Similarly, system member objects cannot be removed: instead, their membership can be suspended through certificate revocation process. An authentication and authorisation mean must be provided by the caller to execute this method.
- **modify_membership**: This method is called to add/remove slice/project membership. In addition, a member role can also be modified using this method call. When requesting member role modification, the caller must provide credentials with sufficient privileges. Additional check must be performed to ensure that method execution would not result in the removal of a member with *LEAD* role who is principal contact for a slice/project.
- **lookup_members, lookup_for_members**: These lookup methods provide public information about the memberships of slice and projects and can be called without providing any credentials. However, the lookup must be performed with given match criterion and results must be filtered according to provided filters.

4.8.1.3 Compatibility with OMNI

Among others APIs, OMNI also supports the Common Federation API; which makes it compatible with the C-BAS clearinghouse. Hence, by configuring some parameters such as the experimenter's certificate & key or the access IP address & port number of C-BAS, the command line interface of OMNI can be used to interact with C-BAS, as well as to setup and execute experiments in FELIX test-beds. This way, OMNI has also served as a testing tool for C-BAS.

4.8.1.4 Compatibility with jFed

jFed is a GUI-based user agent developed within Fed4FIRE project. Like OMNI, jFed also supports the Common Federation API and therefore has full compatibility with C-BAS.

4.8.1.5 Integration with Expedient

Expedient, a GUI-based user agent inherited from FP7 OFELIA, has its own integral standalone clearinghouse. As Expedient contained all authentication and authorisation procedures internally, it offers no interface to interact with a external clearinghouse like C-BAS. Consequently, Expedient has been enhanced within FELIX to enable its communication with C-BAS, as a substitute, external clearinghouse.

As a result, experimenters are facilitated with GUI-based front-ends, Expedient among them; while in the back-end C-BAS methods are called by Expedient to fetch credentials, store/lookup objects like slice, project, keys etc. When interacting with C-BAS, Expedient prints all debug information onto a dedicated log file that can be checked in case of problems. Moreover, front-end of Expedient also displays user friendly error messages if a user action results in an unexpected behaviour.

4.8.1.6 Availability of Clearinghouse module

C-BAS is a software module that provides access to its interface through a Python Flask Server. Being a standalone component, it can be installed on any physical or virtual machine without introducing any particular hardware constraints [23].

The availability of C-BAS access interface is monitored by Expedient and in case of unavailability, an error message is displayed to the experimenters and the island administrator.

4.8.2 Validation procedures

This section explains the validation procedures carried out to verify the correct behaviour of C-BAS.

4.8.2.1 Compliance with Common Federation API

The compliance check of C-BAS with the Common Federation API was performed through *unit tests*, which are part of C-BAS repository. The *unit testing* runs approximately 200 tests to verify various aspects for compliance. For example, it is checked whether:

- All mandatory functions have been implemented
- All implemented functions behave according to the API specifications
- Data type of passed method arguments are as expected
- Information of members, slices, slivers, and projects is correctly stored
- Appropriate error message is returned if a method is called with invalid or wrong number of arguments
- Returned values of all method calls are of correct data type and have expected values
- Proper authentication and authorisation is performed for operation requiring data modification
- Member roles and associated privileges are enforced when performing authorisation
- Privilege delegation works as expected
- C-BAS's internal or user private information is not revealed to the outside world
- Data integrity is maintained after operations involving database modification
- Access interface of C-BAS remains available despite any invalid method calls or internal errors
- CRL is maintained in up-to-date state

The *units tests* are run automatically for each commit to C-BAS repository through Travis CI [24].

4.8.2.2 Correct internal behaviour per method

The correct internal behaviour has been mainly verified through the *unit tests* described above. In addition, we manually checked C-BAS database tables and log files to perform a control on internal behaviour.

4.8.2.3 Integration with Expedient

The validation of Expedient's interface with C-BAS was mainly performed manually. We monitored Expedient and C-BAS error log files and followed several cycles of creating a new member, downloading its certificate and key from Expedient, using aforementioned information to log onto Expedient, creating project, creating slices, accessing slice credential, adding or removing members to project, updating member SSH keys, updating project and slice information, deleting project object. The successful execution of aforementioned tasks validated the interface between C-BAS and Expedient.

4.8.2.4 Compatibility with OMNI

The compatibility of C-BAS with OMNI has been validated through *unit tests*. These automated tests instruct OMNI to execute all those commands where clearinghouse is directly or indirectly involved. These commands include:

- *get_ch_version*
- *listresources*
- *createslice*
- *getslicecred*
- *renewslice*
- *print_slice_expiration*
- *listslices*
- *listslivers*
- *listslicemembers*
- *listprojects*
- *listprojectmembers*
- *listmykeys*

The *unit tests* also check response of these commands for consistency and report any encountered error.

4.8.2.5 Compatibility with jFed

The compatibility of C-BAS with jFed is mainly tested using the *unit test* that come shipped with jFed. These *unit tests*, grouped into Slice Authority Tests and Member Authority Tests, execute a number of Common Federation API calls and analyse return values. These tests include:

1. Slice Authority Tests
    - *getVersion*
    - *getTestUserCredential*
    - *getTestUserInfo*

- *retrieveCredentialSomehow*
- *createProject*
- *findProjectToUseInTests*
- *lookupProjectsByNameNoFilter*
- *createSlice*
- *lookupProjectsByUrnNoFilter*
- *updateProject*
- *getSliceCredentials*
- *lookupProjectsNoFilterAfterUpdate*
- *lookupSlicesNoFilter*
- *updateSlice*
- *lookupProjectMembers*
- *lookupSlicesNoFilterAfterUpdate*
- *lookupProjectsForMember*
- *lookupSliceMembers*
- *modifyProjectMembership*
- *lookupSlicesForMember*
- *modifySliceMembership*
- *lookupSliverInfoNoFilter*

2. Member Authority Tests

- *getVersion*
- *getTestUserCredential*
- *retrieveCredentialSomehow*
- *lookupMemberInfoEmptyFilter*
- *lookupMemberInfoEmptyFilterWithMatchList*
- *lookupMemberInfoNoFilterWithMatchList*
- *lookupMemberInfoNoFilter*
- *lookupPublicMemberInfoFiltered*
- *lookupIdentifyingMemberInfoFilteredWithMatchList*
- *lookupPublicMemberInfoFilteredWithMatchList*
- *lookupIdentifyingMemberInfoFiltered*
- *createKey*
- *lookupKeysNoFilter*
- *updateKey*
- *lookupKeysNoFilterAfterUpdate*
- *deleteKey*
- *lookupKeysNoFilterAfterDelete*

5 Validation Tools

Several tools have been used to validate the developed components. Some are used specifically for the validation of the code or the functionalities while others are used for the deployment and allow the validation as another feature.

5.1 OMNI

The main validation tool to test the functionalities of most of the modules has been OMNI. OMNI is a command line user agent developed within GENI project which allows manually calling GENI commands to an AM and check the received output. By analysing this output, the compliance of AM northbound API with GENIv3 can be tested to assure the correct communication with other modules and federated islands. These commands also allow performing unit tests of each method of the module to check its internal behaviour correctness.

OMNI has been used to test all the modules that must be GENIv3 compatibles (RO, SDNRM, CRM, SERM, TNRM and AAA).

5.2 jFed

jFed [19] is a new GUI tool lately introduced in the FELIX project and presented in [25]. Since jFed is the GUI that experimenters will use to manage FELIX experiments, it has been used as a test tool. FELIX allows editing RSpecs and check the results of running them through different logs and XML responses.

5.3 Public Monitoring

Similar to jFed, the public monitoring system is a tool to allow experimenters check the status of the AMs of the different islands in order to know their availability for the experiment. Once validated that the tool works correctly, the public monitoring system can be, and has been, used as a validation tool since it keeps performs periodic connectivity and login tests and keeps logs of each test. These logs include for each method call its request and response XML RSpecs, making it a very useful tool to check why an RM has failed.

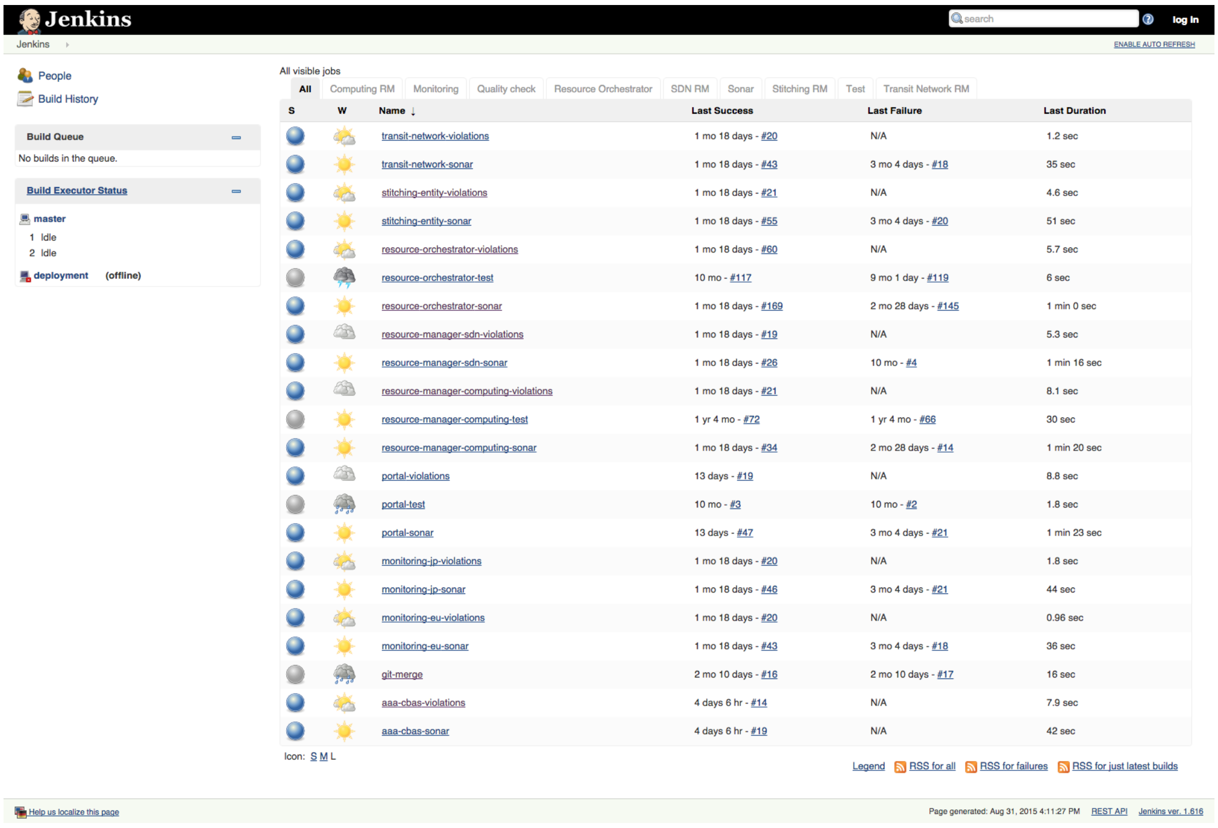
5.4 Jenkins

Although not a validation tool per se, Jenkins [26] is an application that allows automating tasks. In FELIX, we have used it as a *Continuous Integration* server to perform both automating deployment and validation procedures. We have defined automated jobs to evaluate the code and perform some unit testing, and we are working on others such as the automatic merging of different branches to integrate into the master one.

Figure 5.1 shows the main page of Jenkins, showing the public list of tasks with the last build status (pass/fail), the last success and failure (whether the process succeeded or failed, Jenkins keeps a register of the last execution) and finally, the time required by it, which helps detecting stalled processes, either due to some incorrect command or configuration, or a problem in a physical device.

After the Y2 review, the validation tasks were also added to Jenkins. For this, Jenkins was extended with the Pylint plug-in in order to provide extra analysis to that of SonarQube (described below). Here, Pylint checks for errors on the Python code, dead code (imports, unused variables, etc) or deviations from the standard style guide (here, PEP8 [27] is used).

Thanks to its explicit location of the warning, it is easy to address problems. For instance, this is a warning line provided in the Jenkin's *Console Output* after an inspection on code used by RO: *"/opt/felix/resource-orchestrator/modules/resource/orchestrator/src/extensions/geni/pgch.py:31:[E] 'datetime' imported but unused"*. This error corresponds to third-party code, where a Python module is imported but not used. A great number of



Build Queue

No builds in the queue.

Build Executor Status

master
1 Idle
2 Idle
deployment (offline)

All visible jobs

All | Computing RM | Monitoring | Quality check | Resource Orchestrator | SDN RM | Sonar | Stitching RM | Test | Transit Network RM

S	W	Name ↓	Last Success	Last Failure	Last Duration
●		transit-network-violations	1 mo 18 days - #20	N/A	1.2 sec
●		transit-network-sonar	1 mo 18 days - #43	3 mo 4 days - #18	35 sec
●		stitching-entity-violations	1 mo 18 days - #21	N/A	4.6 sec
●		stitching-entity-sonar	1 mo 18 days - #55	3 mo 4 days - #20	51 sec
●		resource-orchestrator-violations	1 mo 18 days - #60	N/A	5.7 sec
●		resource-orchestrator-test	10 mo - #117	9 mo 1 day - #119	6 sec
●		resource-orchestrator-sonar	1 mo 18 days - #169	2 mo 28 days - #145	1 min 0 sec
●		resource-manager-sdn-violations	1 mo 18 days - #19	N/A	5.3 sec
●		resource-manager-sdn-sonar	1 mo 18 days - #26	10 mo - #4	1 min 16 sec
●		resource-manager-computing-violations	1 mo 18 days - #21	N/A	8.1 sec
●		resource-manager-computing-test	1 yr 4 mo - #72	1 yr 4 mo - #66	30 sec
●		resource-manager-computing-sonar	1 mo 18 days - #34	2 mo 28 days - #14	1 min 20 sec
●		portal-violations	13 days - #19	N/A	8.8 sec
●		portal-test	10 mo - #3	10 mo - #2	1.8 sec
●		portal-sonar	13 days - #47	3 mo 4 days - #21	1 min 23 sec
●		monitoring-jp-violations	1 mo 18 days - #20	N/A	1.8 sec
●		monitoring-jp-sonar	1 mo 18 days - #46	3 mo 4 days - #21	44 sec
●		monitoring-eu-violations	1 mo 18 days - #20	N/A	0.96 sec
●		monitoring-eu-sonar	1 mo 18 days - #43	3 mo 4 days - #18	36 sec
●		git-merge	2 mo 10 days - #16	2 mo 10 days - #17	16 sec
●		aaa-cbas-violations	4 days 6 hr - #14	N/A	7.9 sec
●		aaa-cbas-sonar	4 days 6 hr - #19	N/A	42 sec

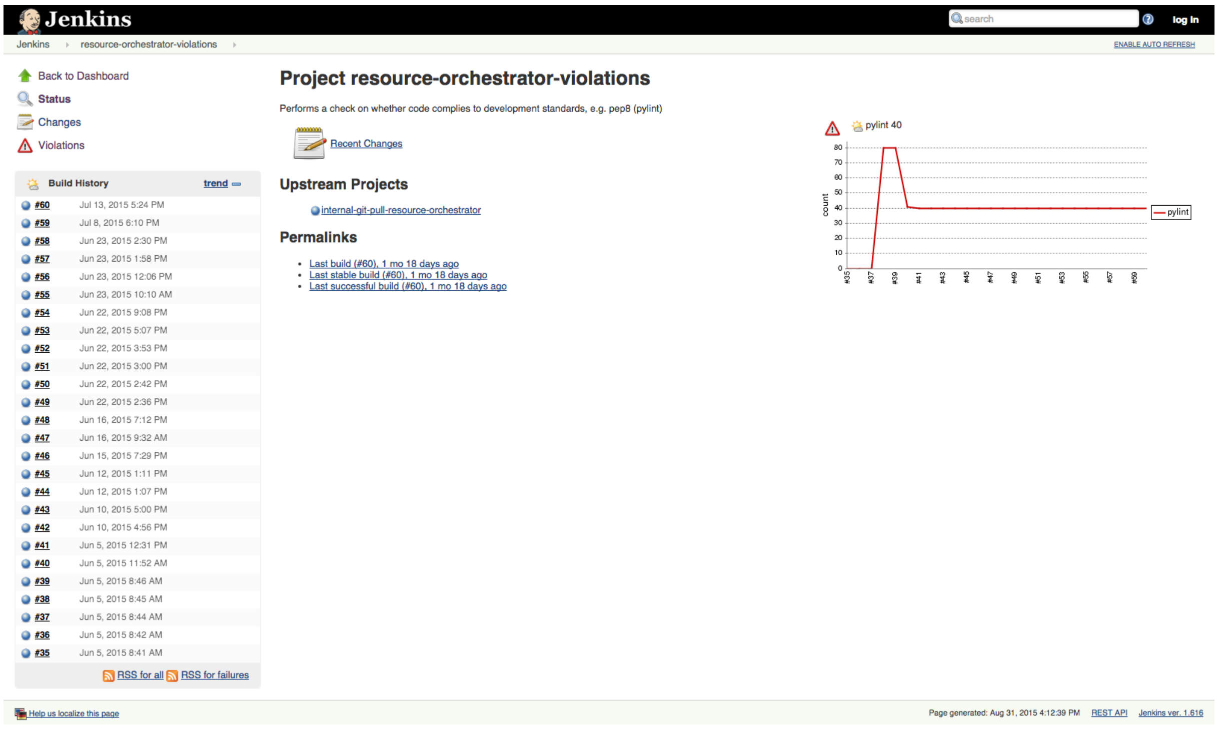
Icon: S M L

Legend RSS for all RSS for failures RSS for just latest builds

Help us localize this page

Page generated: Aug 31, 2015 4:11:27 PM REST API Jenkins ver. 1.616

Figure 5.1: Jenkins main page with list of tasks



Project resource-orchestrator-violations

Performs a check on whether code complies to development standards, e.g. pep8 (pylint)

Recent Changes

Upstream Projects

internal-git-pull-resource-orchestrator

Permalinks

- Last build (#60), 1 mo 18 days ago
- Last stable build (#60), 1 mo 18 days ago
- Last successful build (#60), 1 mo 18 days ago

Build History trend

#60	Jul 13, 2015 5:24 PM
#59	Jul 8, 2015 6:10 PM
#58	Jun 23, 2015 2:30 PM
#57	Jun 23, 2015 1:58 PM
#56	Jun 23, 2015 12:06 PM
#55	Jun 23, 2015 10:10 AM
#54	Jun 22, 2015 9:08 PM
#53	Jun 22, 2015 5:07 PM
#52	Jun 22, 2015 3:53 PM
#51	Jun 22, 2015 3:00 PM
#50	Jun 22, 2015 2:42 PM
#49	Jun 22, 2015 2:36 PM
#48	Jun 16, 2015 7:12 PM
#47	Jun 16, 2015 9:32 AM
#46	Jun 15, 2015 7:29 PM
#45	Jun 12, 2015 1:11 PM
#44	Jun 12, 2015 1:07 PM
#43	Jun 10, 2015 5:00 PM
#42	Jun 10, 2015 4:56 PM
#41	Jun 5, 2015 12:31 PM
#40	Jun 5, 2015 11:52 AM
#39	Jun 5, 2015 8:46 AM
#38	Jun 5, 2015 8:45 AM
#37	Jun 5, 2015 8:44 AM
#36	Jun 5, 2015 8:42 AM
#35	Jun 5, 2015 8:41 AM

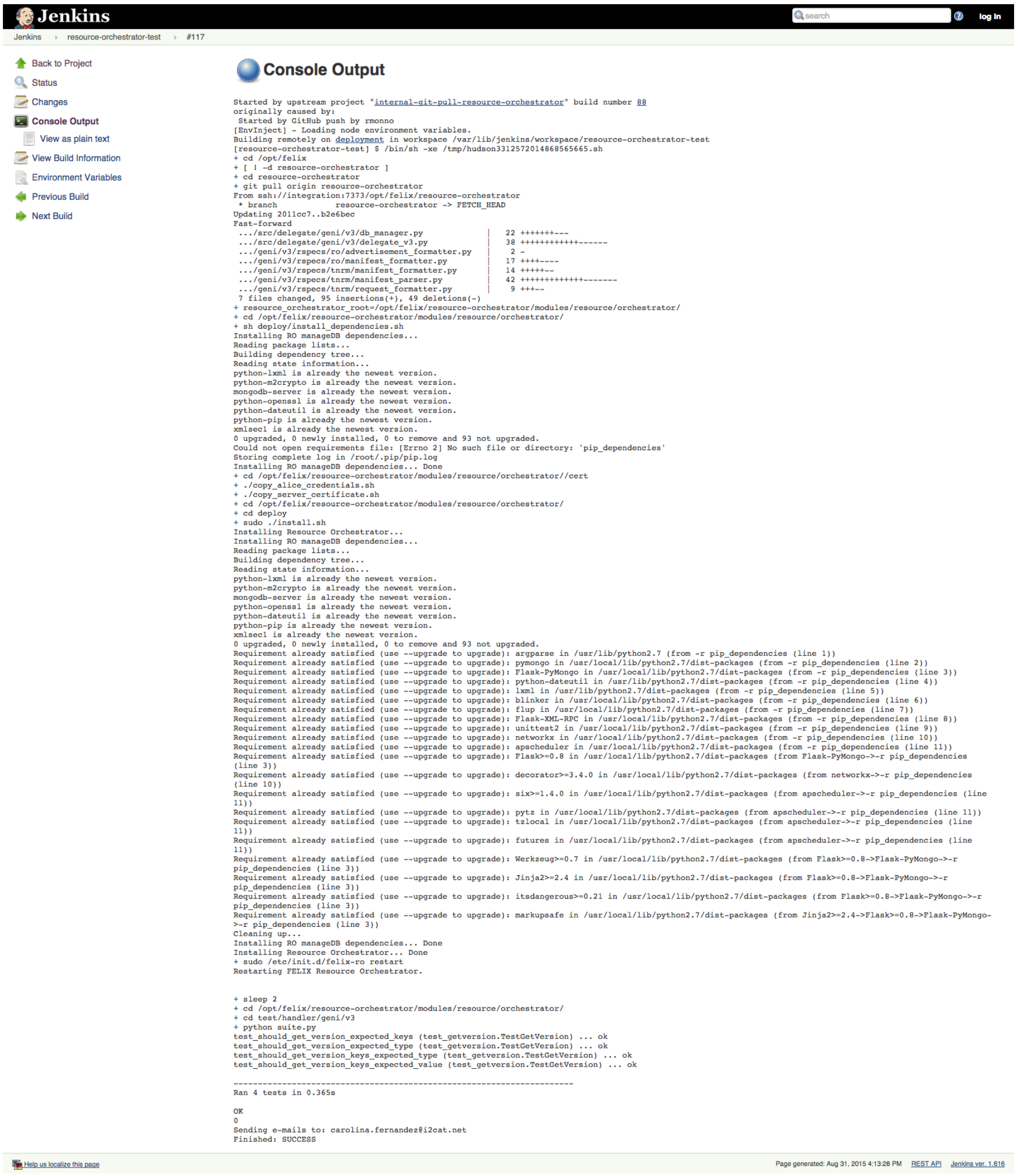
RSS for all RSS for failures

Help us localize this page

Page generated: Aug 31, 2015 4:12:39 PM REST API Jenkins ver. 1.616

Figure 5.2: Jenkins page for validation of RO code, using Pylint

the violations reported by Pylint were addressed after Y2's review, as it can be observed in the chart on the right side of Figure 5.2.



```
Started by upstream project "internal-git-pull-resource-orchestrator" build number 88
originally caused by:
 Started by GitHub push by rmonno
[EnvInject] - Loading node environment variables.
Building remotely on deployment in workspace /var/lib/jenkins/workspace/resource-orchestrator-test
[resource-orchestrator-test] $ /bin/sh -xe /tmp/hudson3312572014868565665.sh
+ cd /opt/felix
+ [ ! -d resource-orchestrator ]
+ cd resource-orchestrator
+ git pull origin resource-orchestrator
From ssh://integration:7373/opt/felix/resource-orchestrator
 * branch            resource-orchestrator -> FETCH_HEAD
Updating 2011cc7..b2e6bec
Fast-forward
 .../src/delegate/geni/v3/db_manager.py              |   22 +++++++---
 .../src/delegate/geni/v3/delegate_v3.py             |   38 ++++++++++++------
 .../geni/v3/rspecs/ro/advertisement_formatter.py    |    2 -
 .../geni/v3/rspecs/ro/manifest_formatter.py         |   17 ++++----
 .../geni/v3/rspecs/tnrm/manifest_formatter.py       |   14 +++++--
 .../geni/v3/rspecs/tnrm/manifest_parser.py          |   42 +++++++++++++-------
 .../geni/v3/rspecs/tnrm/request_formatter.py        |    9 +++--
 7 files changed, 95 insertions(+), 49 deletions(-)
+ resource_orchestrator_root=/opt/felix/resource-orchestrator/modules/resource/orchestrator/
+ cd /opt/felix/resource-orchestrator/modules/resource/orchestrator/
+ sh deploy/install_dependencies.sh
Installing RO manageDB dependencies...
Reading package lists...
Building dependency tree...
Reading state information...
python-lxml is already the newest version.
python-m2crypto is already the newest version.
mongodb-server is already the newest version.
python-openssl is already the newest version.
python-dateutil is already the newest version.
python-pip is already the newest version.
xmlsec1 is already the newest version.
0 upgraded, 0 newly installed, 0 to remove and 93 not upgraded.
Could not open requirements file: [Errno 2] No such file or directory: 'pip_dependencies'
Storing complete log in /root/.pip/pip.log
Installing RO manageDB dependencies... Done
+ cd /opt/felix/resource-orchestrator/modules/resource/orchestrator//cert
+ ./copy_alice_credentials.sh
+ ./copy_server_certificate.sh
+ cd /opt/felix/resource-orchestrator/modules/resource/orchestrator/
+ cd deploy
+ sudo ./install.sh
Installing Resource Orchestrator...
Installing RO manageDB dependencies...
Reading package lists...
Building dependency tree...
Reading state information...
python-lxml is already the newest version.
python-m2crypto is already the newest version.
mongodb-server is already the newest version.
python-openssl is already the newest version.
python-dateutil is already the newest version.
python-pip is already the newest version.
xmlsec1 is already the newest version.
0 upgraded, 0 newly installed, 0 to remove and 93 not upgraded.
Requirement already satisfied (use --upgrade to upgrade): argparse in /usr/lib/python2.7 (from -r pip_dependencies (line 1))
Requirement already satisfied (use --upgrade to upgrade): pymongo in /usr/local/lib/python2.7/dist-packages (from -r pip_dependencies (line 2))
Requirement already satisfied (use --upgrade to upgrade): Flask-PyMongo in /usr/local/lib/python2.7/dist-packages (from -r pip_dependencies (line 3))
Requirement already satisfied (use --upgrade to upgrade): python-dateutil in /usr/lib/python2.7/dist-packages (from -r pip_dependencies (line 4))
Requirement already satisfied (use --upgrade to upgrade): lxml in /usr/lib/python2.7/dist-packages (from -r pip_dependencies (line 5))
Requirement already satisfied (use --upgrade to upgrade): blinker in /usr/local/lib/python2.7/dist-packages (from -r pip_dependencies (line 6))
Requirement already satisfied (use --upgrade to upgrade): flup in /usr/local/lib/python2.7/dist-packages (from -r pip_dependencies (line 7))
Requirement already satisfied (use --upgrade to upgrade): Flask-XML-RPC in /usr/local/lib/python2.7/dist-packages (from -r pip_dependencies (line 8))
Requirement already satisfied (use --upgrade to upgrade): unittest2 in /usr/local/lib/python2.7/dist-packages (from -r pip_dependencies (line 9))
Requirement already satisfied (use --upgrade to upgrade): networkx in /usr/local/lib/python2.7/dist-packages (from -r pip_dependencies (line 10))
Requirement already satisfied (use --upgrade to upgrade): apscheduler in /usr/local/lib/python2.7/dist-packages (from -r pip_dependencies (line 11))
Requirement already satisfied (use --upgrade to upgrade): Flask>=0.8 in /usr/local/lib/python2.7/dist-packages (from Flask-PyMongo->-r pip_dependencies
(line 3))
Requirement already satisfied (use --upgrade to upgrade): decorator>=3.4.0 in /usr/local/lib/python2.7/dist-packages (from networkx->-r pip_dependencies
(line 10))
Requirement already satisfied (use --upgrade to upgrade): six>=1.4.0 in /usr/local/lib/python2.7/dist-packages (from apscheduler->-r pip_dependencies (line
11))
Requirement already satisfied (use --upgrade to upgrade): pytz in /usr/local/lib/python2.7/dist-packages (from apscheduler->-r pip_dependencies (line 11))
Requirement already satisfied (use --upgrade to upgrade): tzlocal in /usr/local/lib/python2.7/dist-packages (from apscheduler->-r pip_dependencies (line
11))
Requirement already satisfied (use --upgrade to upgrade): futures in /usr/local/lib/python2.7/dist-packages (from apscheduler->-r pip_dependencies (line
11))
Requirement already satisfied (use --upgrade to upgrade): Werkzeug>=0.7 in /usr/local/lib/python2.7/dist-packages (from Flask>=0.8->Flask-PyMongo->-r
pip_dependencies (line 3))
Requirement already satisfied (use --upgrade to upgrade): Jinja2>=2.4 in /usr/local/lib/python2.7/dist-packages (from Flask>=0.8->Flask-PyMongo->-r
pip_dependencies (line 3))
Requirement already satisfied (use --upgrade to upgrade): itsdangerous>=0.21 in /usr/local/lib/python2.7/dist-packages (from Flask>=0.8->Flask-PyMongo->-r
pip_dependencies (line 3))
Requirement already satisfied (use --upgrade to upgrade): markupsafe in /usr/local/lib/python2.7/dist-packages (from Jinja2>=2.4->Flask>=0.8->Flask-PyMongo-
>-r pip_dependencies (line 3))
Cleaning up...
Installing RO manageDB dependencies... Done
Installing Resource Orchestrator... Done
+ sudo /etc/init.d/felix-ro restart
Restarting FELIX Resource Orchestrator.


+ sleep 2
+ cd /opt/felix/resource-orchestrator/modules/resource/orchestrator/
+ cd test/handler/geni/v3
+ python suite.py
test_should_get_version_expected_keys (test_getversion.TestGetVersion) ... ok
test_should_get_version_expected_type (test_getversion.TestGetVersion) ... ok
test_should_get_version_keys_expected_type (test_getversion.TestGetVersion) ... ok
test_should_get_version_keys_expected_value (test_getversion.TestGetVersion) ... ok

----------------------------------------------------------------------
Ran 4 tests in 0.365s

OK
0
Sending e-mails to: carolina.fernandez@i2cat.net
Finished: SUCCESS
```

Help us localize this page
Page generated: Aug 31, 2015 4:13:28 PM
REST API
Jenkins ver. 1.616

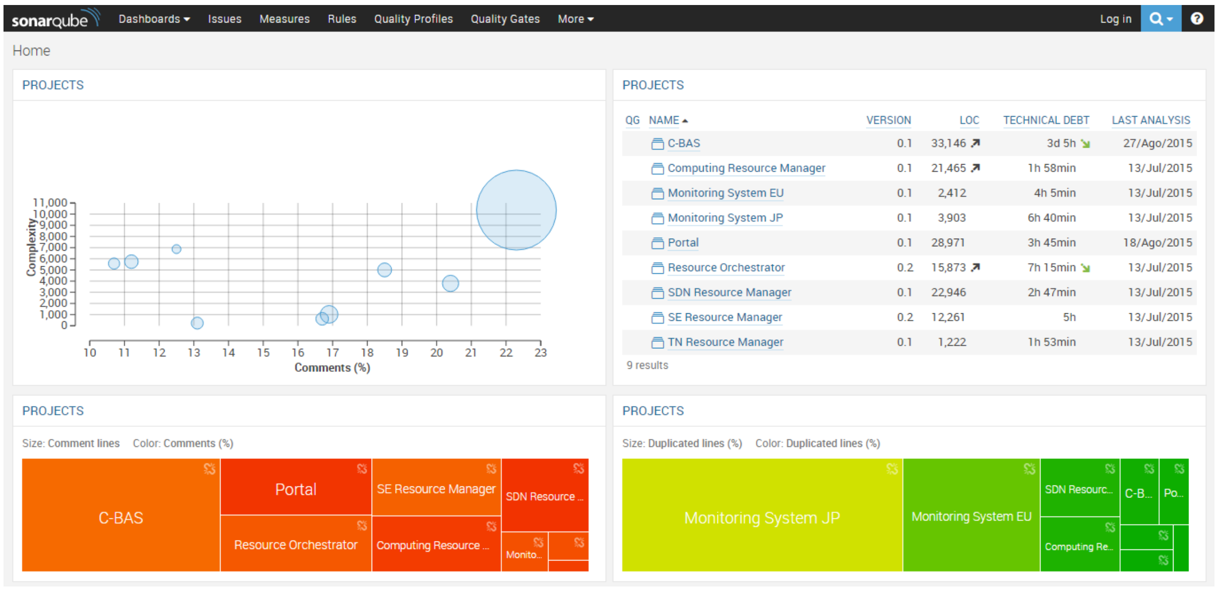
Figure 5.3: Jenkins page of an early deployment and unit test on RO

Finally, we have also used Jenkins to perform automatic deployment after every new push to the shared repository, and then to run unit tests. After some time, whoerver, we disabled this task, as the unit testing we were performing was already carried out through other tools (Fed4FIRE's FLS monitor). In any case, Figure 5.3 shows an screenshot of an early deployment of the RO and one of the unit tests on its northbound API, which is now performed in a different way.

5.5 SonarQube

Though the main aim of the FELIX validation stages is on checking the validity of the functionalities, we have also dedicated some effort on keeping the quality and simplicity on the code so as to minimise future errors due to unexpected behaviour or difficulty of maintenance.

In this sense, SonarQube [28] is an open platform to keep an eye on the code quality (e.g., compliance to standards, e.g. PEP8, and the code simplicity, or even its branch coverage). In FELIX, an instance of SonarQube has been installed to check several aspects of code quality: complexity, duplications of lines, comments, coding rules, etc.



PROJECTS

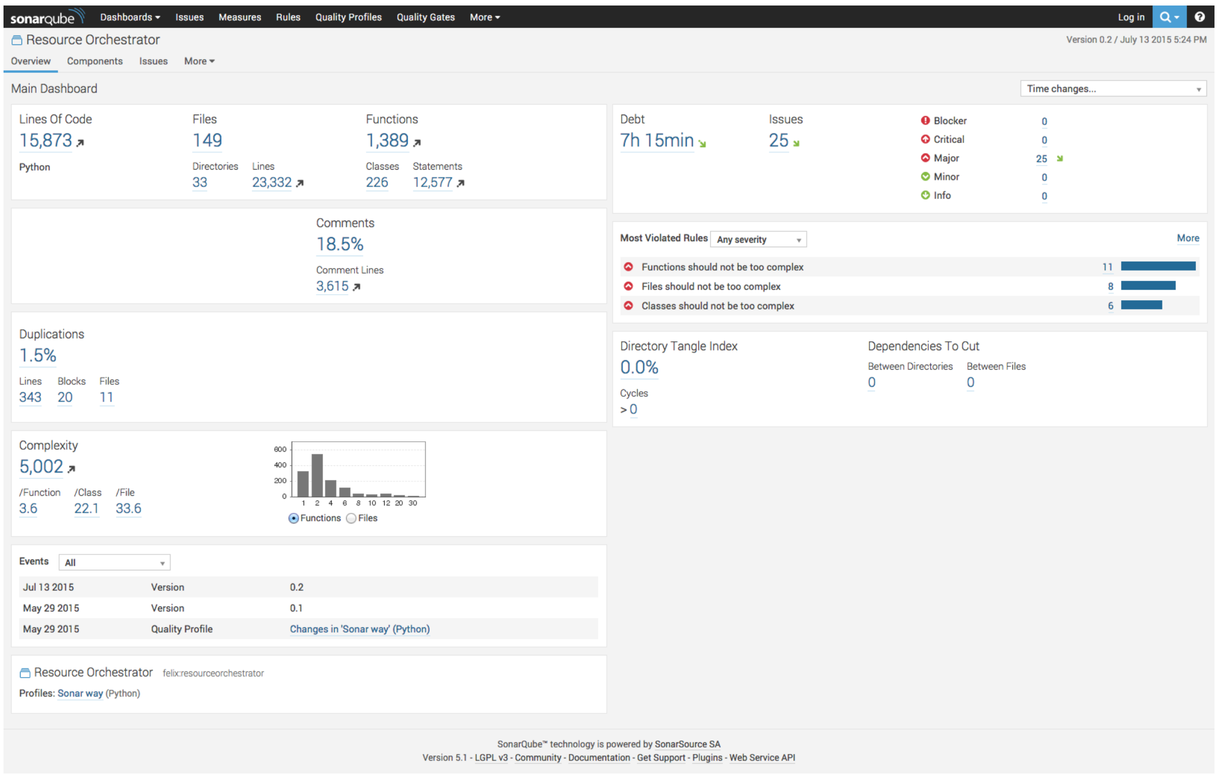
QG	NAME	VERSION	LOC	TECHNICAL DEBT	LAST ANALYSIS
	C-BAS	0.1	33,146	3d 5h	27/Ago/2015
	Computing Resource Manager	0.1	21,465	1h 58min	13/Jul/2015
	Monitoring System EU	0.1	2,412	4h 5min	13/Jul/2015
	Monitoring System JP	0.1	3,903	6h 40min	13/Jul/2015
	Portal	0.1	28,971	3h 45min	18/Ago/2015
	Resource Orchestrator	0.2	15,873	7h 15min	13/Jul/2015
	SDN Resource Manager	0.1	22,946	2h 47min	13/Jul/2015
	SE Resource Manager	0.2	12,261	5h	13/Jul/2015
	TN Resource Manager	0.1	1,222	1h 53min	13/Jul/2015

9 results

Figure 5.4: SonarQube main page

Through proper and manual configuration of each project (in this case, we have set up a project per FELIX software module, though sometimes there may be more than one project per module), SonarQube can be triggered to perform inspection on the FELIX source. The triggering is automatically performed through Jenkins, upon code is committed. The main page of the tool, with graphical diagrams and charts on no. classes, issues or the list of the available projects, can be seen in Figure 5.4.

After Y2 review, the list of projects in SonarQube was also increased, as well as the list of metrics to be shown per project and the configuration per project was improved to only account for the code developed within the FELIX project and draw a specific line for improvements. Thereafter, some of the reported issues have been addressed either through the analysis performed by tasks in Jenkins or SonarQube. Figure 5.5 shows the project page for the RO source code, featuring the number of issues and the estimated technical debt to solve them; an analysis of the complexity per Python function, class and file; a percentage on the duplicated lines.



Resource Orchestrator

Version 0.2 / July 13 2015 5:24 PM

Main Dashboard

Lines Of Code 15,873
Python

Files 149
Directories 33
Lines 23,332

Functions 1,389
Classes 226
Statements 12,577

Comments 18.5%
Comment Lines 3,615

Duplications 1.5%
Lines 343
Blocks 20
Files 11

Complexity 5,002
/Function 3.6
/Class 22.1
/File 33.6

Debt 7h 15min

Issues 25

Blocker	0
Critical	0
Major	25
Minor	0
Info	0

Most Violated Rules — More

Functions should not be too complex	11
Files should not be too complex	8
Classes should not be too complex	6

Directory Tangle Index 0.0%
Cycles > 0

Dependencies To Cut
Between Directories 0
Between Files 0

Events

Jul 13 2015	Version	0.2
May 29 2015	Version	0.1
May 29 2015	Quality Profile	Changes in 'Sonar way' (Python)

Resource Orchestrator felix:resourceorchestrator
Profiles: Sonar way (Python)

SonarQube™ technology is powered by SonarSource SA
Version 5.1 - LGPL v3 - Community - Documentation - Get Support - Plugins - Web Service API

Figure 5.5: SonarQube page with analysis on the RO source

6 Conclusions and Summary

In order to provide a reliable set of components, the FELIX project has adopted a set of measures to deploy and validate the functionalities and intercommunication of each component. Methodologies and tools help to this task.

In this document we presented details on the status of the deployed FELIX components at the different islands or domains, and how their functionalities were successfully tested. Summing up, the main test for the different Resource Managers was their compliance with GENIv3 and the correct internal behaviour. For these tasks, the OMNI CLI and jFed (and, therefore also F4F-FLS) were the main tools used for both manual and automatic testing via GENIv3 calls.

Finally, other interesting validation procedures that have been performed in parallel to development refer to the assessment of the code quality, as it is an attempt to produce coherent, easy to debug and long-term lasting code. The main validation tools used to integrate and check code quality were the open source tools called Jenkins and SonarQube, which have been briefly described in the last sections of this document.

References

[1] ``Ryu SDN framework.'' http://osrg.github.io/ryu/.

[2] ``POX OpenFlow Controller.'' http://www.noxrepo.org/pox/about-pox/.

[3] R. Krzywania, et al., ``FELIX Deliverable D2.2, General Architecture and Functional Blocks,'' tech. rep.

[4] ``GENI Aggregate Manager API v3.'' http://groups.geni.net/geni/wiki/GAPI_AM_API_V3.

[5] ``Fed4FIRE First-Level Support for FELIX testbeds.'' http://flsmonitor.fed4fire.eu/fls.html?testbedcategory=felix.

[6] R. Monno, et al., ``FELIX Deliverable D3.1, Resource Planning and Provisioning,'' tech. rep.

[7] ``FELIX Public Repository.'' https://github.com/ict-felix.

[8] ``MongoDB.'' https://www.mongodb.com/collateral/mongodb-30-whats-new.

[9] ``FELIX tests for SDNRM module.'' https://github.com/dana-i2cat/felix/tree/ocf/optin_manager/src/python/openflow/optin_manager/geni/v3/tests.

[10] ``FELIX tests for CRM module.'' https://github.com/dana-i2cat/felix/tree/ocf/vt_manager/src/python/vt_manager/communication/geni/v3/tests.

[11] ``GENI (Global Environment for Network Innovations) - website.'' http://www.geni.net.

[12] ``GENI Network Stitching - Overview.'' https://wiki.maxgigapop.net/twiki/pub/GENI/NetworkStitching/geni-network-stitching-architecture-overview.pdf.

[13] ``eiSoil Framework.'' https://github.com/EICT/eiSoil.

[14] C. Bermudo, et al., ``FELIX Deliverable D3.4, End User Tools and API,'' tech. rep.

[15] T. Kudoh, et al., ``Network services interface: An interface for requesting dynamic inter-datacenter networks,'' *Optical Fiber Communication Conference (OFC)*, Mar. 2013.

[16] T. Ikeda, et al., ``FELIX Deliverable D3.2, Slice Monitoring,'' tech. rep.

[17] ``Common Federation API.'' http://groups.geni.net/geni/wiki/ CommonFederationAPIv2, Nov. 2013.

[18] ``The Omni client.'' http://trac.gpolab.bbn.com/gcf/wiki/Omni, 2015.

[19] ``Java-based framework for testbed federation.'' http://jfed.iminds.be/, 2015.

[20] J. Naous, et al., ``Expedient: A centralized pluggable clearinghouse to manage geni experiments,'' Jan. 2010.

[21] ``GENI: Global Environment for Network Innovations.'' http://www.geni.net.

[22] ``Fed4FIRE Projects.'' http://www.fed4fire.eu/, 2015.

[23] U. Toseef, et al., ``Implementation of C-BAS: Certificate-based AAA for SDN Experimental Facilities,'' in *Proc. NCCA 2015*, June 2015.

[24] ``Travis CI: Continuous Integration and Deployment.'' https://travis-ci.org/EICT/C-BAS, 2015.

[25] C. Fernandez, et al., ``FELIX Deliverable R2.1, FELIX Recommendation 1, year 2,'' FELIX Deliverable R2.1, Jul. 2015.

[26] ``Jenkins.'' https://jenkins-ci.org/.

[27] ``PEP 0008 -- Style Guide for Python Code.'' https://www.python.org/dev/peps/pep-0008/.

[28] ``SonarQube.'' http://www.sonarqube.org/.

Appendix A: Software Stack Deployment per Island

This section provides further technical details and status information for every FELIX software module deployed across the different FELIX domains. For each island, the table include the software components deployed and its location (address, port and endpoint) as well as other details as the Aggregate Manager (AM) type and version or the Last change.

This information is complemented with pictures of the topology per domain, available at the following address: http://www.ict-felix.eu/?page_id=332

A.1 PSNC domain

	Protocol	Host	IP	Port	Endpoint	AM type	AM version	Last change
Expedient	https	Dell#1	10.216.65.235	443	-	-	-	8-May-2015
Clearinghouse	https	Dell#1	10.216.65.235	8000	-	-	-	8-May-2015
C-BAS (AAA)	https	Dell#1	10.216.65.235	8008	-	-	-	8-May-2015
Resource Orchestrator	https	Dell#1	10.216.65.235	8440	xmlrpc/geni/3/	geni	3	8-May-2015
GENI Opt-In (SDNRM)	https	Dell#1	10.216.65.235	8443	xmlrpc/geni/3/	geni	3	8-May-2015
VT manager (CRM)	https	Dell#1	10.216.65.235	8445	xmlrpc/geni/3/	geni	3	8-May-2015
SERM	https	Dell#1	10.216.65.235	8447	xmlrpc/geni/3/	geni	3	8-May-2015
MS	http	Dell#1	10.216.65.235	8449	monitoring-system/ monitoring/	-	-	13-May-2015

Table A.1: Software modules installed in PSNC domain

A.2 i2CAT domain

	Protocol	Host	IP	Port	Endpoint	AM type	AM version	Last change
Expedient	https	deployment.felix.i2cat.net	84.88.40.167	443	-	-	-	16-May-2015
C-BAS (AAA)	https	deployment.felix.i2cat.net	84.88.40.167	8008	-	-	-	28-Jan-2015
Resource Orchestrator (Master)	https	deployment.felix.i2cat.net	84.88.40.167	18440	xmlrpc/geni/3/	geni	3	16-May-2015
Resource Orchestrator	https	deployment.felix.i2cat.net	84.88.40.167	8440	xmlrpc/geni/3/	geni	3	16-May-2015
SDNRM	https	exp.i2cat.fp7-ofelia.eu	10.216.12.4	8443	xmlrpc/geni/3/	geni	3	16-May-2015
CRM	https	exp.i2cat.fp7-ofelia.eu	10.216.12.4	8445	xmlrpc/geni/3/	geni	3	16-May-2015
SERM	https	deployment.felix.i2cat.net	84.88.40.167	8447	xmlrpc/geni/3/	geni	3	11-May-2015
MS	http	deployment.felix.i2cat.net	84.88.40.167	8448	monitoring-system/monitoring/	-	-	11-May-2015

Table A.2: Software modules installed in i2CAT domain

A.3 iMinds domain

	Protocol	Host	IP	Port	Endpoint	AM type	AM version	Last change
FOAM *(SDNRM)*	https	foam.atlantis. ugent.be		3626	foam/gapi/2	geni	2	-
VirtualWall *(CRM+SERM)*	https	www.wall2.ilabt. iminds.be		12369	protogeni/xmlrpc/ am/3.0	geni	3	-
Clearinghouse	https	www.wall2.ilabt. iminds.be		12369	protogeni/xmlrpc/ sa	protogeni	1	-

Table A.3: Software modules installed in iMinds domain

A.4 EICT domain

	Protocol	Host	IP	Port	Endpoint	AM type	AM version	Last change
Expedient	https	exp.eisland.fp7-ofelia.eu	10.216.123.5	443	-	-	-	16-May-2015
C-BAS (AAA)	https	exp.eisland.fp7-ofelia.eu	10.216.123.5	8008	-	-	-	17-Mar-2015
Resource Orchestrator	https	exp.eisland.fp7-ofelia.eu	10.216.123.5	8440	xmlrpc/geni/3/	geni	3	18-Mar-2015
SDNRM	https	exp.eisland.fp7-ofelia.eu	10.216.123.5	8443	xmlrpc/geni/3/	geni	3	16-May-2015
CRM	https	exp.eisland.fp7-ofelia.eu	10.216.123.5	8445	xmlrpc/geni/3/	geni	3	16-May-2015
SERM	https	exp.eisland.fp7-ofelia.eu	10.216.123.5	8447	xmlrpc/geni/3/	geni	3	in progress

Table A.4: Software modules installed in EICT domain

A.5 KDDI domain

	Protocol	Host	IP	Port	Endpoint	AM type	AM version	Last change
Expedient	https	felix-kddi.dcn.jgn-x.jp	10.216.68.5/ 202.180.38.151	443	-	-	-	14-Apr-2015
Resource Orchestrator	https	felix-kddi.dcn.jgn-x.jp	10.216.68.5/ 202.180.38.151	8440	xmlrpc/geni/3/	geni	-	14-Apr-2015
SDNRM	https	felix-kddi.dcn.jgn-x.jp	10.216.68.5/ 202.180.38.151	8443	xmlrpc/geni/3/	geni	3	14-Apr-2015
CRM	https	felix-kddi.dcn.jgn-x.jp	10.216.68.5/ 202.180.38.151	-	-	-	-	not available
SERM	https	felix-kddi.dcn.jgn-x.jp	10.216.68.5/ 202.180.38.151	8447	xmlrpc/geni/3/	geni	3	21-Apr-2015
MMS-JP	http	felix-mms.dcn.jgn-x.jp	10.216.68.6/ 202.180.38.153	8448	monitoring-system/monitoring/	-	-	14-Apr-2015
MS-JP	http	felix-ms.dcn.jgn-x.jp	10.216.68.7/ 202.180.38.154	8448	monitoring-system/monitoring/	-	-	14-Apr-2015

Table A.5: Software modules installed in KDDI domain

A.6 AIST domain

A.6.1 Island no. 1

	Protocol	IP	Port	Endpoint	AM type	AM version	Last change
Resource Orchestrator (Master)	https	10.216.69.141/ 163.220.30.141	18440	xmlrpc/geni/3/	geni	3	31-July-2015
Resource Orchestrator	https	10.216.69.137/ 163.220.30.137	8440	xmlrpc/geni/3/	geni	3	15-May-2015
SDNRM	https	10.216.69.244/ 163.220.30.148	8443	xmlrpc/geni/3/	geni	3	15-May-2015
KVM-CRM	https	10.216.69.137/ 163.220.30.137	8445	xmlrpc/geni/3/	geni	3	15-May-2015
TNRM	https	10.216.69.243/ 163.220.30.147	8446	None or RPC2	geni	3	15-May-2015
SERM	https	10.216.69.244/ 163.220.30.148	8447	xmlrpc/geni/3/	geni	3	15-May-2015
MS-JP	http	10.216.69.139/ 163.220.30.139	8448	monitoring-system/monitoring/	-	-	15-May-2015

Table A.6: Software modules installed in AIST island no. 1

A.6.2 Island no. 2

	Protocol	IP	Port	Endpoint	AM type	AM version	Last change
Resource Orchestrator	https	10.216.70.138/ 163.220.30.138	8440	xmlrpc/geni/3/	geni	3	31-July-2015
SDNRM	https	10.216.70.144/ 163.220.30.144	8443	xmlrpc/geni/3/	geni	3	31-July-2015
KVM-CRM	https	10.216.70.138/ 163.220.30.138	8445	xmlrpc/geni/3/	geni	3	31-July-2015
SERM	https	10.216.70.144/ 163.220.30.144	8447	xmlrpc/geni/3/	geni	3	31-July-2015
MS-JP	http	--	8448	monitoring-system/monitoring/	-	-	-

Table A.7: Software modules installed in AIST island no. 2

The scientific/academic work is financed from financial resources for science in the years 2013 - 2016 granted for the realization of the international project co-financed by Polish Ministry of Science and Higher Education.