


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

D2.4 – Second federation architecture

Work package	WP2
Task	T2.1
Due date	30/11/2013
Submission date	05/03/2014
Deliverable lead	Brecht Vermeulen (iMinds)
Version	Final
Authors	All WP2 partners
Reviewers	Felicia Lobillo Vilela (ATOS) Ally Hume (EPCC)

Abstract	This document is the second deliverable of the architectural task of the Fed4FIRE project. It defines the architecture that will be implemented during the project's second development cycle. It refines and extends the architecture defined in the first development cycle, taking as input requirements from WP3 (Infrastructure community), WP4 (Service community), WP7 (Trustworthiness), WP8 (First Level Support) and task 2.3 (sustainability). Practical experience gained through the development of the architecture of the first development cycle has also been taken into account.
Keywords	Architecture, SFA, experiment lifecycle, federation

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

Disclaimer

The information, documentation and figures available in this deliverable, is written by the Fed4FIRE (Federation for FIRE) *– project consortium under EC co-financing contract FP7-ICT-318389 and does not necessarily reflect the views of the European Commission. The European Commission is not liable for any use that may be made of the information contained herein.*

Executive Summary

This second deliverable of task 2.1 defines the architecture for cycle 2 of the Fed4FIRE project.
It takes as input requirements of WP3 (Infrastructure community), WP4 (service community), WP8 (First Level Support), SLA management and task 2.3 (sustainability) and defines an architecture that copes with as many requirements as possible and will be implemented for cycle 2 of Fed4FIRE. It is an evolution based on the cycle 1 architecture.

For cycle 2, just as in cycle 1, discovery, specification and provisioning will be done based on the SFA GENI AM API v3 standard, while for advanced reservation and extended policy based authorization extensions are currently being made. Regarding the RSpecs, in cycle 2 each testbed can still provide its own RSpec to be used and tools have to be adapted to these RSpecs.

The architecture defines also multiple identity providers with a chain of trust, and a central portal accompanied with an identity provider and directories (machine and human readable) for tools, testbeds and certificates.

For First Level Support, just as in cycle 1 the architecture defines facility monitoring which should be identical for all testbeds and should make it possible for first level support to have a high level overview of testbed and resource availability.

For monitoring, three types of monitoring have been identified and described. Facility monitoring is used to monitor a testbed as a whole and is e.g. interesting for First Level Support and experimenters to know if a testbed is functioning correctly or not. Infrastructure monitoring is monitoring information provided by the testbed itself to an experimenter (e.g. switch port statistics, spectrum measurements or virtualization infrastructure monitoring) which is normally not available to an experimenter. Experimenter measuring is related to monitoring activities that the experimenter can do on the nodes themselves.

For SLA management, a first architecture is introduced which is based on monitoring information coming from the testbeds. This should make it possible to gain experience with simple SLA evaluation based on resource availability during an experiment.

As a general conclusion, this 2nd cycle architecture is an evolvement of cycle 1, with many improvements, clarifications and additions on multiple points, but the basic ideas remain the same, which means that implementation and deployment do not have to change radically but will be further go to unified interfaces and workflows.

Acronyms and Abbreviations

AM	Aggregate Manager
API	Application Programming Interface
CPU	Central Processing Unit
FLS	First Level Support
FRCP	Federated Resource Control Protocol
IMS	IP Multimedia System
NEPI	Network Experiment Programming Interface
NIC	Network Interface Controller
OMF	cOntrol and Management Framework
OML	Open Measurement Library
PI	Principal Investigator
RAM	Random-access Memory
REST	Representational State Transfer
RSpec	Resource Specification
SFA	Slice-based Federation Architecture
SFI	Command line SFA client (meaning of acronym is not documented)
SLA	Service Level Agreement
SOAP	Simple Object Access Protocol
SSH	Secure Shell
VPN	Virtual Private Network
XMPP	Extensible Messaging and Presence Protocol
URN	Unified Resource Name
PI	Principal Investigator
URL	Uniform Resource Locator
SOAP	Simple Object Access Protocol
REST	Representational State Transfer

Table of Contents

List of figures

1 Introduction

The Fed4FIRE project is structured around three development cycles (Figure 1). This deliverable is the second in the architectural task of the project, and describes the architecture for the second development cycle. It is an evolution of the architecture that was defined in cycle 1. Based on practical experience with that architecture, and on the updated requirements listed by WP3, WP4, WP7 and WP8, this second federation architecture includes both revisions of architectural components already supported by the first architecture, and also introduces some entirely new concepts. In a similar manner, this second architecture will then be further revised in cycle 3.

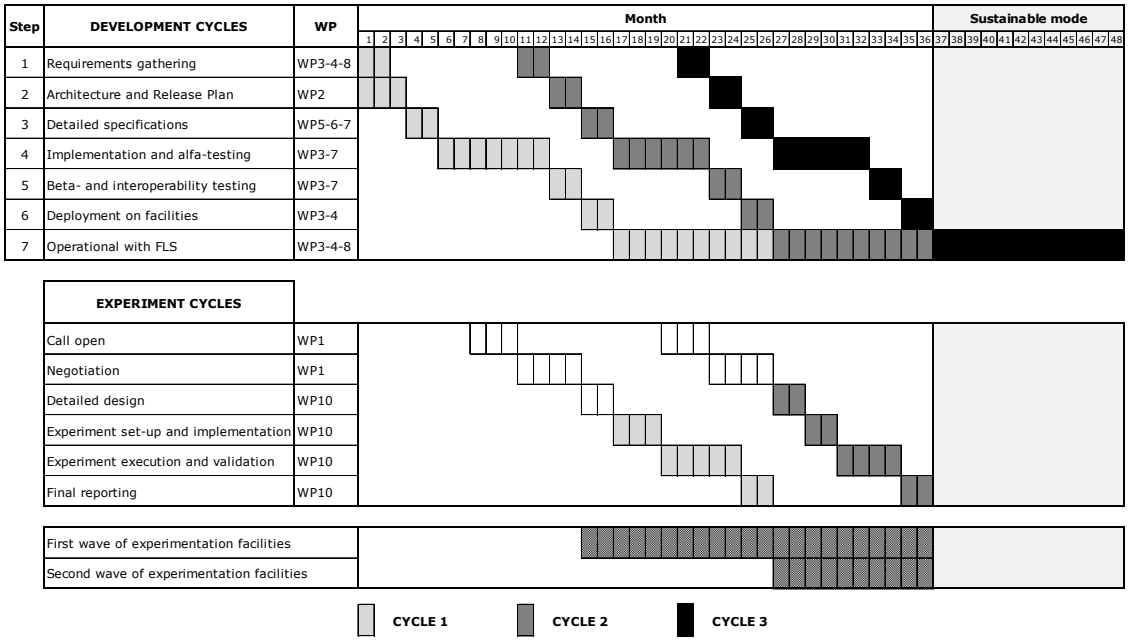


Figure 1: Overview of the three development cycles within Fed4FIRE

Whenever an experimenter performs an experiment, he or she performs a set of sequential actions that will help him or her to execute an experiment idea on a testbed and obtain the desired results. These different actions are considered to be part of what is called “the experimental lifecycle”, which can be fragmented in different functions. Because the federation architecture presented in this deliverable is closely related to these different steps of that experimental lifecycle, the Fed4FIRE definition of the experiment lifecycle is reiterated below. Note that compared to the previous version of this deliverable (D2.1 – First federation architecture [1]), the notion of facility monitoring, infrastructure monitoring and experiment measurement has been slightly refined.

Function		Description
Resource discovery		Finding available resources across all testbeds, and acquiring the necessary information to match required specifications.
Resource specification		Specification of the resources required during the experiment, including compute, network, storage and software libraries. E.g., 5 compute nodes, 100Mbps links, specific network topology, 1 TB storage node, 1 IMS server, 5 measurement agents.
Resource reservation		Allocation of a time slot in which exclusive access and control of particular resources is granted.
Resource provisioning	Direct (API)	Instantiation of specific resources directly through the testbed API, responsibility of the experimenter to select individual resources.
	Orchestrated	Instantiation of resources through a functional component, which automatically chooses resources that best fit the experimenter's requirements. E.g., the experimenter requests 10 dual-core machines with video screens and 5 temperature sensors.
Experiment control		Control of the testbed resources and experimenter scripts during experiment execution. This could be predefined interactions and commands to be executed on resources (events at startup or during experiment workflow). Examples are: startup or shutdown of compute nodes, change in wireless transmission frequency, instantiation of software components during the experiment and breaking a link at a certain time in the experiment. Real-time interactions that depend on unpredictable events during the execution of the experiment are also considered.
Monitoring	Facility monitoring	Instrumentation of resources to supervise the behavior and performance of testbeds, allowing system administrators or first level support operators to verify that testbeds are performing correctly.
	Infrastructure monitoring	Instrumentation by the testbed itself of resources to collect data on the behavior and performance of services, technologies, and protocols. This allows the experimenter to obtain monitoring information about the used resources that the experimenter could not collect himself. An example of such infrastructure monitoring is the provisioning by the testbed of information regarding the CPU load and NIC congestion on the physical host of a virtual machine resource. The experimenter can only collect monitoring data on the level of the VM, but the testbed provides infrastructure monitoring capabilities that make this data available to the experimenter.
Measuring	Experiment measuring	Collection of experimental data generated by frameworks or services that the experimenter can deploy on its own.
Permanent storage		Storage of experiment related information beyond the experiment lifetime, such as experiment description, disk images and measurements.

Function	Description
Resource release	Release of experiment resources after deletion or expiration the experiment.

This remainder of this deliverable is structured as follows:

- Section 2 describes the inputs and requirements that lead to this architecture definition of cycle 2. The inputs and requirements come from:
  - WP2 (D2.1): the first federation architecture [1]
  - WP3 (D3.2): Infrastructure community [2]
  - WP4 (D4.2): Services community [3]
  - WP8 (D8.4): First Level Support [4]
  - WP7: SLA management
  - WP2 (D2.3): Sustainability [5]
- Section 3 describes the architecture itself, together with basic concepts regarding the Fed4FIRE context. It is described in 4 parts:
  - Resource discovery, resource reservation and resource provisioning
  - Monitoring and measurement
  - Experiment control
  - SLA management
- Section 4 describes the differences between the cycle 1 and cycle 2 architecture.
- Section 5 touches back on the requirements to evaluate how many requirements are tackled by the architecture.
- Section 6 concludes the deliverable
- The following appendices are attached:
  - Appendix A: Requirements from the infrastructure community (D3.2)
  - Appendix B: Requirements from the services community (D4.2)
  - Appendix C: Requirements from the SLA management
  - Appendix D: Interaction between Policy Decision Point and Aggregate Manager
  - Appendix E: What information does a user credential and slice credential contain
  - Appendix F: Subauthorities in a member and slice authority
  - Appendix G: Evaluation of possible architectural approaches for SLA management
  - Appendix H: Workflow for registering an account and getting a certificate
  - Appendix I: Workflow for creating an SSH key on different operating systems
  - Appendix J: Run a first experiment

2 Overview of inputs

This section will describe all inputs leading to the architectural choice for cycle 2. As stated before, these are the architectural design of the first cycle, the requirements defined in WP3, WP4, WP7 and WP8 and the project's task on sustainability.

2.1 First federation architecture

When defining the first federation architecture in D2.1 [1], the project evaluated 4 different types of architectures. Based on that evaluation, the architecture for cycle 1 was defined for the different steps in the experiment lifecycle. For cycle 1 it was decided that resource discovery, specification and provisioning would be supported based on the SFA GENI AM API v3 standard, while for advanced reservation and extended policy based authorization extensions would be foreseen. Regarding the resource specification, the aim was to get them as unified as possible in cycle 1 (with GENI RSpec v3 as a guideline), while for cycles 2 and 3 a unified ontology based RSpec definition was identified as the target.

That architecture also defined multiple identity providers with a chain of trust, and a central portal accompanied with an identity provider and directories (machine and human readable) for tools, testbeds and certificates. These services function as 'broker services'.

For First Level Support, the architecture defined facility monitoring which should be identical for all testbeds and should make it possible for first level support to have a high level overview of testbed and resource availability. Infrastructure monitoring, experiment measurement and experiment control were not yet standardized within WP2's federation architecture for cycle 1. This was however perceived not to be a blocking issue, as the experimenters can deploy their own tools for this and they are not dependent on the testbed providers. In the consequent development step, being the establishment of more detailed specifications by the vertical work packages 5, 6 and 7 (responsible for the development of the federation framework), it was then decided that at least the Federated Resource Control Protocol (FRCP) API was to be adopted for experiment control, and OML for experiment measurement.

The first cycle federation architecture was designed to cover as many requirements as possible. However, since it was not feasible to cover all of them, there were several high priority requirements that were reserved for later cycles. From these, most important non-tackled requirements were considered to be the ones regarding inter-connectivity between testbeds and storage of all kinds of information. It was concluded that both of them would be tackled in cycles 2 and 3 of Fed4FIRE. Some other requirements were identified as being only partly resolved, but were expected to be fully resolved when the ontology based RSpecs/resource descriptions will be deployed in cycles 2 and 3. These specific conclusions should be taken into account when designing the architecture for cycle 2.

The general conclusion of D2.1 was however that the first cycle architecture and way forward already coped with a large number of the requirements that had been set by the other work packages. The practical experience gained with the development of this first federation architecture confirmed the statement above: today it is still considered to be a solid basis for the Fed4FIRE federation, and it is not needed to profoundly restructure it. Only refinements, clarifications and additions of some specific new functionalities are needed.

2.2 Requirements from the infrastructures community (D3.2)

The purpose of the document, "D3.2 Infrastructure community federation requirements, version 2" [2], was to gather (for the second time) requirements from the Infrastructure community's perspective in order to build a federation of FIRE facilities. Compared to the first iteration, this deliverable is characterized by a more profound outreach to the FIRE community. The following sources were applied during the requirements elicitation process:

- Requirements not covered or partially covered in cycle 1.
- Interactions with specific research communities related to the WP3 testbeds
- Analysis of the set of proposals submitted to Fed4FIRE for experimentation in the project's 1st Open Call willing to experiment on WP3 testbeds. These proposals represent the experiments that are actually envisaged by FIRE experimenters, and contained very valuable direct and indirect information regarding their requirements.
- New non-trivial use cases that were specifically written for D3.2 that were designed to push the envisaged federation tools to their limits in terms of scale and heterogeneity.

Compared to the first iteration of WP3's requirements deliverable, D3.2 was also characterized by a more transparent prioritization process. This revised prioritisation mechanism has been applied to identify the high-priority requirements by weighting the importance at the federation level for the use cases and open calls analysed. These requirements are considered essential and their implementation should be included in the second cycle developments as much as possible and as long as this is feasible in terms of other constraints (e.g. effort). They are listed in Appendix A: Requirements from the infrastructures community (D3.2). They can be summarized as follows:

- Fed4FIRE must provide a clear view on what **node capabilities** are available, and this should be defined in the same way across the federation. It should be possible for the experimenter to **select the resources** he/she would like to include in the experiment.
- Resource discovery must be integrated into uniform tools through **federation-wide APIs**.
- For nodes that have wired and/or wireless network connections to other nodes within the same testbed, it should be possible to identify the physical **topology.** It should also be known how different infrastructures are/can be **interconnected** (e.g. via layer 3 or layer 2).
- Fed4FIRE must provide **hard reservations** of the available resources in a fair, secure and fully automated manner.
- APIs are required to enable **direct instantiation** of both physical and virtualized resources for experiments.
- Fed4Fire must provide the ability to **install a specific custom Linux kernel or distribution on the nodes.** Fed4FIRE must provide the possibility to access a node as root user. Software installation through a packet manager (e.g., apt-get) must be possible. It should be possible to create a binary image of the entire hard disk drive once a node has been fully installed.
- It should be possible to already define what the specific resources should do at startup (install additional software, copy specific files, start specific daemons/tools).
- **Nodes must be accessible via SSH.**
- It must be possible to describe **advanced experiment scenarios by the use of a script that will be executed by a control engine**. This engine should be general enough to support the control of all possible kinds of Future Internet technology.
- Fed4FIRE must provide an **easy way for experimenters to store measures during the experiment runtime for later analysis.**

- **Common characteristics should be stored automatically** during an experiment (CPU load, free RAM, Tx/Rx errors, wireless interference, etc.)
- **Monitoring info regarding the state of the resources should be opened up to all experimenters**, and not only those that were using the resources. This way they can choose the best resources for their experiments.
- **As less overhead as possible should be expected from the monitoring and measurement support frameworks.**
- The user must be able to request **on-demand measurements**. In order to do so, they will need to express that they want agents with such on-demand polling capacities.
- **Access to the data should be properly secured.**
- Experiment configurations should be stored in order to **replay experiments** and compare results of different runs.
- Fed4FIRE must provide the mean of accessing all testbeds within the federation using **one single account** (username/password). Fed4FIRE should also provide authentication by the use of public SSH keys. Access to the Fed4FIRE APIs (discovery, reservation, provisioning, etc.) should also be protected by an authentication mechanism.
- It should be **easy for new experimenters without any affiliation to the federation to create their Fed4FIRE identity.**
- The resources within a Fed4FIRE infrastructure should be able to reach the resources deployed in the other Fed4FIRE infrastructures through a layer 3 Internet connection. **Interconnectivity solutions** should not introduce unneeded complexity in the experiment such as VPN or other tunnels. The ability to conduct **IPv6** measurements and to interact with the nodes of other testbeds over IPv6 should be enabled.
- Per experiment, Fed4FIRE should provide the possibility to r**eserve bandwidth on the links that interconnect specific infrastructures**.

2.3 Requirements from the services community (D4.2)

The purpose of the deliverable "D4.2 Second input from the Services and Applications community to the architecture" [3] was to gather the second set of requirements from the Services and Applications community's perspective in order to continue the construction of federation of FIRE facilities. In this context, requirements were gathered from different sources:

- Requirements not covered or partially covered in cycle 1.
- During the development of the first cycle, the services and application community identified the need for a greater orientation of Fed4FIRE towards the concept of "application services". From this community's perspective, the second iteration of the Fed4FIRE architecture should provide further utilities or software available for the experimenters in order to ease their interaction with the federated resources, which will contribute to sustainability and reusability, allowing experimenters to take advantage of all the resources and services available in the federation even with limited knowledge concerning the underlying technologies. This work was carried out through interviews with all testbeds within the project at the end of cycle 1 and it intended to strengthen the exposure and use of applications within the federation.
- New use cases based on experiments that could run on the infrastructure and services provided by the Fed4FIRE testbeds belonging to the Services and Applications community at the beginning of cycle 2 (i.e. BonFIRE, FuSeCo and SmartSantander). Different partners provided these use cases as an exercise to join different testbeds from the services and applications perspective before the proposals of the open calls were available. This exercise was influenced and inspired by the knowledge of their specific domains of expertise (i.e.

Cloud Computing, LTE services and Smart Cities) and they represent typical experiments these communities would carry out. As for the FI-PPP, although the new wave of vertical use cases was at an early stage by then, the interaction of Fed4FIRE with these projects also produced a scenario which, inspired in a FI-PPP vertical use case, could be built over current FIRE facilities.

- The practical experience and lessons learned of the BonFIRE community was also identified as a source of requirements for this cycle. The idea was to ease its integration in the federation in such a way that most of its full functionality can be exposed through Fed4FIRE architecture and tools.
- Feedback from the community stakeholders, mainly taking into consideration the proposals presented to Fed4FIRE for experimentation in the 1st Open Call, which represent real examples of the kind of experiments that could be run on the federated testbeds.

These different sources produced a set of requirements that gather needs for the 2nd cycle of Fed4FIRE from the Services and Applications community's perspective. Once requirements had been grouped, the prioritisation was made according to how many proposals and use cases benefit from a requirement. This exercise has resulted in some valuable insights. Important requirements in cycle 2 are, for example, those related to the services perspective and the ease of use in general but also interconnectivity among different testbeds. These requirements are considered essential and their implementation should be included in the second cycle developments much as possible and as long as this is feasible in terms of other constraints (e.g. effort). The actual requirements are listed in Appendix B: Requirements from the services community (D4.2). They can be summarized as follows:

- Fed4FIRE should provide a **standardised way of registering and unregistering external devices** (any device that is not part of the testbed itself, but that is to be included in the experiment, such as dedicated smartphone, tablet, novel product, etc.)
- During an experiment, the experimenter might need to decide if and how to change the amount of computing or network resources allocated to a service.
- **Resources must be described in a homogeneous manner** so that the experimenter can compare the experiments in different testbeds.
- The experimenter should have multiple **resource reservation** options.
- Some **features** could be **executed automatically**. For example, a reserved experiment could start automatically.
- Multiple testbeds of different kinds should be federated together in one experiment through a **set of common tools.**
- **Fed4FIRE tools should be user friendly to the experimenter**.
- It should be possible to **pack up resources in services** offered to the experimenter so that he/she is isolated from the real infrastructure.
- Fed4FIRE must **provide the means for experimenters and third parties to develop and/or deploy applications on top of Fed4FIRE infrastructure.**
- **Experimenters should be able to manage resources they have created** (share/unshare/destroy).
- Fed4FIRE must provide tools to create, view, update, and terminate **monitoring configurations** related to shared resource types or experiments in real time.
- Fed4FIRE should enable the experimenter accessing all resources (hardware and software) with own privileges by performing login only once at the start of the experiment (**single sign on**).

- Fed4FIRE will make the distinction between requests of local users, PhD students from other institutes (research), students (practical exercises), in order to know what kind of experimenter is logging in and from where and apply **policies** accordingly.
- **Interconnections between testbeds** should be in place (WAN links/public Internet access) so that during an experiment, experimenters can invoke services inside and outside a testbed, moving data between testbeds without it going through his personal machine.

2.4 Requirements from First Level Support (D8.4)

First Level Support (FLS) is one of the central functions of the Fed4FIRE federation. It provides a common facility for the logging and resolution of faults. It provides a direct line of communications with experimenters and with the support functions in the individual testbeds. Deliverable "D8.4 Second input to WP2 concerning first level support" [4] considered how the initial architectural requirements of FLS have been met and described the processes that FLS intends to use in the operational phase of Fed4FIRE. It considered how, in a live operational environment within the federation, operational support is likely to function. Given the absence of any operational data today, it used, as a proxy, experience gained from a similar federated operational environment in the sector of backbone networks. Using this analysis a number of proposals were made for the way in which the operational architecture of the federation can be developed. Some of these proposals implied specific requirements on the project's federation architecture. They are reiterated bellow.

Organize pro-active announcements of maintenance

It is expected that maintenance will represent a significant element of the operational issues affecting the federation. It is vital that such maintenance, which is usually scheduled, should be registered in advance in the FLS Trouble Ticket System (TTS). Systematically doing so, will avoid outages caused by maintenance being diagnosed as incidents. **Automated mechanisms that allow testbeds to easily communicate scheduled maintenance to the FLS as part of their operations is an important architectural requirement.** Even where maintenance is unscheduled, logging the maintenance in the TTS directly prior to starting work, will be effective. All testbed operational functions shall have an account on the TTS and can log in and raise the necessary ticket.

Establish Appropriate Subject Matter Expert links

D8.4 indicated that for GÉANT, approximately half of the trouble tickets opened are associated with pre-planned operational work, the other half being associated with spontaneously occurring technical problems. In the former case, FLS will need to interact with operational staff within the testbed, whereas, in the latter case, the interaction is equally likely to be with developers, either in the project, or in a specific testbed. Both testbed operational staff and developers are Subject Matter Experts (SME). But there is a distinct difference between their skill sets even though, in some cases, they may be one and the same person. In order to effectively develop appropriate lines of communication FLS would need to develop contacts with the two types of SME identified above. In particular, for testbeds, this will require the identification of an appropriate operational support person or function to act as an 'operational' SME for that test-bed. **There is a need to develop mechanisms to allow testbeds to automatically provide and maintain information about the identity of their operational SMEs.**

2.5 Requirements from SLA management

Service Level Agreements (SLA) represent a contractual relationship between a service consumer and a service provider. In the case of Fed4FIRE, this relationship needs to be established between the experimenter and the testbeds. The need for SLAs has been touched in the requirement deliverables of WP3 [2] and WP4 [3], but the corresponding implications on the architecture have not been studied there in a profound level of detail. Therefore WP7 decided to further investigate these SLA related architectural requirements more profoundly. Since there is no specific deliverable planned in WP7 regarding architectural requirements, it was decided to present the results of that WP7 study in this section of the architectural deliverable. So in contradiction to the previous sections, the requirements from SLA management describe new requirements for the architecture that have never been described in any Fed4FIRE deliverable before.

SLAs are used when capability is outsourced to a 3rd party provider as a mechanism to increase trust in providers by encoding dependability commitments and ensuring the level of Quality of Service is maintained to an acceptable level. SLA management also provides information for later accounting, depending on the terms and conditions gathered in the SLA and on whether this SLA has been met by all parties or not. For example, this can determine that a certain penalty or reward is applied to one of the parties. SLAs describe the service that is delivered, its properties and the obligations of each party involved. Moreover, SLAs establish that in case the guarantee is fulfilled or violated, rewards or penalties –monetary or not– can be applied, respectively.

In Fed4FIRE there are several providers (testbeds) involved in an experiment and SLAs must be agreed for all of those who adopt SLAs.

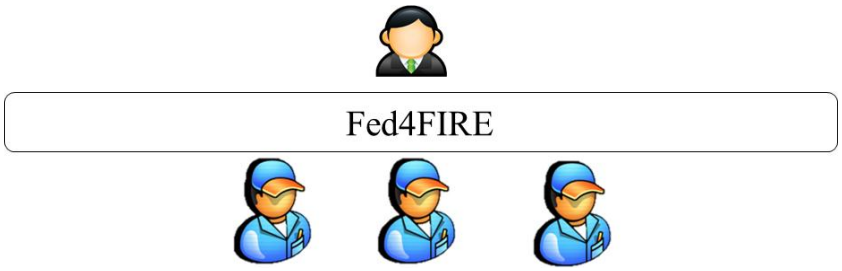


Figure 2: In Fed4FIRE, the experimenter is the customer and the testbeds are the service providers

Besides, testbeds are heterogeneous and different resources and services can be offered within one testbed. The following diagram represents this heterogeneity within one experiment:

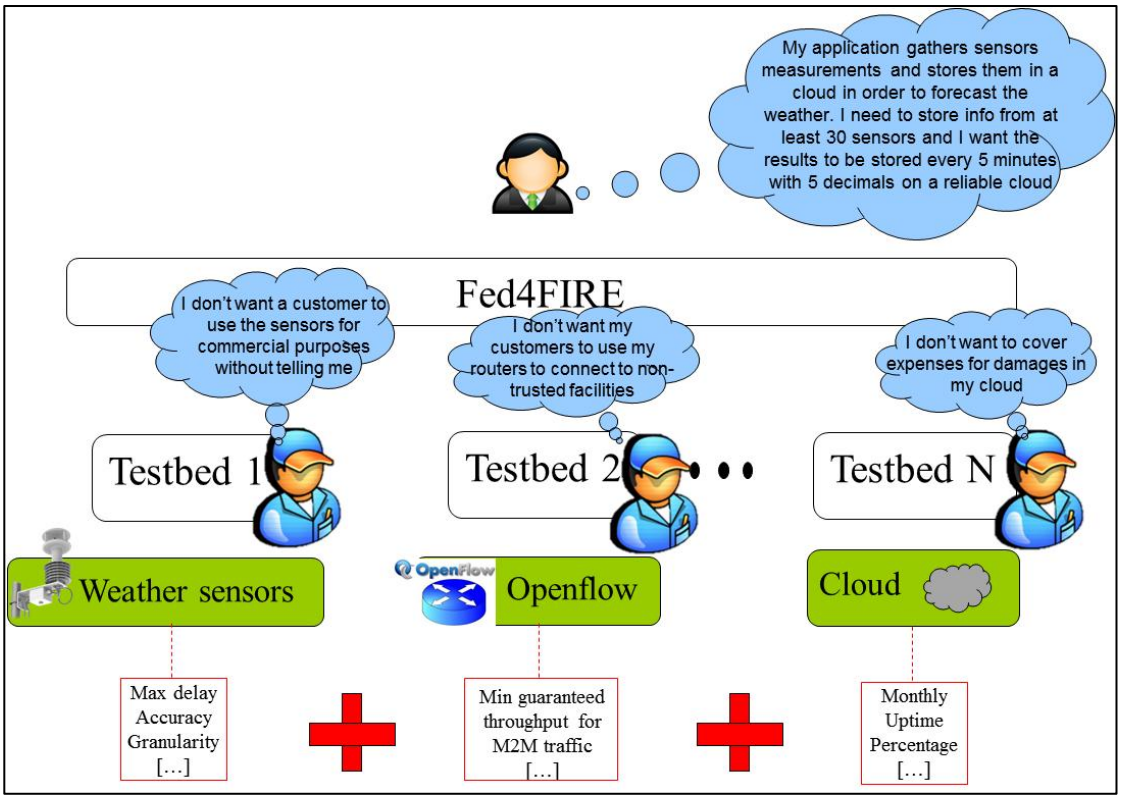


Figure 3: Example of the complexity of SLAs in the Fed4FIRE context

Due to this complexity and to the fact that most testbeds at the beginning of the Fed4FIRE project had no SLA mechanisms in place, the first steps towards implementing SLA management in Fed4FIRE focused on understanding the nature of potential commitments among the different service providers. For this, interviews were organized with every testbed of the project to discuss their views on SLAs. From these interviews, WP7 was able to derive the specific SLA-related architectural requirements. These should be considered when designing the project's architecture for cycle 2. This is explained in more detail in Appendix C Requirements from SLA management. They can be summarized as follows:

- It should be possibly to classify experimenters into **different profiles**, and offer them specific SLAs based on that profile (possibly for a fee).
- The SLA management system should help testbeds retrieve intelligent information concerning the **testbed usage**. It should provide the means for the testbed to expose a certain SLA according to the available capacity at a given moment.
- The **SLA management should be aware of reservations** for those testbeds who operate on this basis.
- When an experiment goes wrong (time out, etc.), it is essential to distinguish whether the problem is originated in the infrastructure (**SLA not met**) or if this is caused by the experimenter's own software or data-set used during the experiment.
- Fed4FIRE must provide the means for infrastructure providers to **publish offerings**, experimenter requirements in terms of QoS and browse/compare offerings. There should be a mechanism for a **negotiation** to agree SLA conditions between the experimenter and each provider.
- Once the user begins the experiment, the SLA management should be able to compare each individual SLA with the monitoring of every testbed involved. In case the SLA is not met, the

SLA management system must be able to clearly identify and make visible which testbed(s) did and did not commit.

- In case experimenters do not meet their obligations, the SLA management must facilitate the claim for some compensation by all affected testbeds who require so. The other way around, users should also be **informed of SLA violations**.
- Fed4FIRE must provide the means to manage and monitor **aggregated SLAs** by joining partial SLAs.
- Fed4FIRE must provide the means for every experimenter and testbed provider to be aware and **sign the terms and conditions** of the federation
- SLA management should provide machine readable information in order for the **monitoring to know what parameters to measure in order to fulfil the SLA** and anticipate upcoming failures.
- The **SLA management should provide information to the central reputation component** in Fed4FIRE.
- In Fed4FIRE, there are low level resource services (e.g. testbeds offer raw resources) but also high-level application services (that run over the resources). Fed4FIRE must be able to deal with SLAs for both kinds of services.
- The SLA management in Fed4FIRE should be **open and based on market standards**

2.6 Requirements from a sustainability point of view (D2.3)

In this phase of the project, Task 2.3 regarding sustainability has delivered its first sustainability plan (D2.3) [5]. This deliverable has presented the methodology that will be used in the remainder of the sustainability task within the Fed4FIRE project. Starting from the value proposition, five business scenarios have been defined, ranging from the "Invisible Coordination" up to the "Integrator" scenario. In order to analyze the scenarios in a structured way, the deliverable defined an evaluation checklist. The final goal was to present a business plan for the future federation, by evaluating the proposed business scenarios and indicating the most realistic strategy. This methodology is thus the main outcome of D2.3, and will be applied and refined in the upcoming deliverables regarding sustainability.

So at this time of the project, the sustainability task has defined its research methodology in detail, and will make detailed statements regarding financial sustainability and its corresponding requirements towards the federation architecture in its next deliverable "D2.6 Second Sustainability Plan", which is due in July 2014. However, at this moment in time the sustainability task T2.3 could already confirm that that the sustainability considerations defined in the project's first architectural deliverable (D2.1) are still valid today. This confirmation is based on the task's experience gained through the finalization of its first sustainability plan (D2.1), and through the ongoing activities that will result in the second sustainability plan (D2.6). Furthermore, that same experience allowed to further refine these sustainability requirements on the federation architecture. These refined statements are given below:

- From a sustainability point of view, it is preferential that the number of required central components is minimal. The reason to **prefer an architecture that defines as less 'central' components as possible** is that these components put a high risk on the federation in terms of scalability and long-term operability. The basic principle of this approach is that every architectural function (e.g. authority, tool, etc.) can and should be implemented by multiple instantiations. So, e.g. if an authority disappears, you can still get the needed credentials at another one. This means also that the federation allows per definition heterogeneity and it still works with the bare minimum of an experimenter, a tool and a testbed. And this is really

attractive from the sustainability viewpoint as it is in line with each testbed's implementation, which also starts with an experimenter, a tool and a testbed. This basic principle implies that the most important things of a federation are open and standardized APIs that are adopted by tools and testbeds, and the establishment of trust relationships between the testbeds belonging to the federation. In order to maximize the potential of this approach in terms of sustainability, these standards should be defined in collaboration with relevant international communities outside of the Fed4FIRE project. This way, tools will be able to easily change who they talk to (e.g. authority or testbed), without being confronted with interoperability issues, even outside the Fed4FIRE-specific context.

- It is also **required that the federation framework supports the joining and leaving of testbeds very easily**, as this is expected to be common practice.
- **It should be easy to add new tools, while the dependency on specific tools or components for the federation should be minimized**. This avoids that the end of support of a specific tool makes the federation unusable
- Finally, it is also **required that the experimenters can join and leave the federation easily**, that there is some notion of **delegation** (to make it more scalable for lots of experimenters) and that PIs (**principal investigators**) can put an end time on experimenters (e.g. students) or can withdraw experiments they have approved.

3 Architecture for Fed4FIRE development cycle 2

3.1 Introduction

The previous section provided an overview of the requirements and constraints imposed on the design of the Fed4FIRE federation architecture for cycle 2. One of the identified basic principles is that the Fed4FIRE architecture should be able to cope with heterogeneous testbed software frameworks, with a focus on adopting the same (standardized) federating interfaces on top of the existing frameworks. This way tools can work with multiple testbeds, orchestration engines should only cope with a single type of interface, and user accounts can be shared over the testbeds. These basic principles were already answered by the architecture of cycle 1, and will be strengthened even more in cycle 2.

To tackle these requirements, and the many other inputs specified in section 2, the federation architecture that is defined in this deliverable encompasses a rather large amount of architectural components. Therefore the detailed architecture discussion has been split up in multiple parts:

- Legend of the architectural diagrams
- Resource discovery, resource reservation and resource provisioning
- Monitoring and measurement
- Experiment control
- SLA management

Before diving into the detail of the second Fed4FIRE federation architecture, a last remark that should be made is that during the design process, careful consideration was given to possible alignment with the work in GENI (http://www.geni.net) in the US. As a result, the second Fed4FIRE federation architecture does not only meet the requirements and other inputs defined by the project in section 2, but it is also interoperable with GENI.

3.2 Legend of the architectural diagrams

Although the architecture is presented in multiple parts, we intend to make these different parts as uniform as possible. As will become clear later in this chapter, this was achieved by visualizing the different architectural parts in the same manner. This section gives some more details about the different elements that are always depicted on the diagrams.

3.2.1 Concept of federations and actors

Before introducing the diagrams of the project's federation architecture, it is important to clarify what we exactly understand by the concept of federation:

- In a federation, testbeds, services, experimenters and authorities have a common trust. E.g. testbeds trust the authorities in the federation (they trust that the certificates which are signed by these authorities contain correct information and thus identity).
- However, this says nothing about policies (who can do what in the federation), only that we trust the entities in the federation.
- A testbed, service, authority or experimenter can be part of multiple federations.
- A federation can be as small as two single testbeds who have a mutual agreement.

- Some federations can be rather short lived, so it should be easy to set them up and tear them down.

Some examples of federations that are in line with the above concept are of course the Fed4FIRE federation, but also the Planetlab federation (Planetlab Europe and Planetlab Central), the Emulab federation (federation based on all testbeds with the same control framework Emulab) and the GENI federation (US testbeds, experimenters and authorities in the GENI project). The existence of these (and other) testbed federations clearly illustrates the need for allowing testbeds, services, authorities and experimenters to be part of multiple federations. E.g. the Planetlab Europe testbed is among others part of the Fed4FIRE and Planetlab federation. The Virtual Wall testbed is among others part of the Fed4FIRE and Emulab federation.

Taking the above interpretation of a federated environment into account, a specific format to depict the project's second federation architecture could be established. In every architectural diagram, the same columns are shown vertically (see Figure 4). Each of them corresponds with a specific actor in the federation:

- **Actors inside the federation**:
  - **Testbeds** with **resources** (=nodes, e.g. a server, a WiFi node, etc)
  - **Experimenters** who belong to the federation if they have credentials which are acknowledged by the federation partners
  - **Federator**: central components which ease the federation, e.g. with directory services. It has to be stressed that in the Fed4FIRE approach these federation components are NOT strictly necessary for operating (where operation means experimenters which do experiments on testbeds). These central components are just there for the convenience of both the testbeds and the experimenters.
- **Actors outside the federation**: the architectural design takes into consideration that there are testbeds, experimenters and authorities also outside of the federation. Testbeds inside the federation can choose to trust some of them if they want. This means that Fed4FIRE testbeds have the freedom to grant access to their testbeds to such actors outside of the federation, or they can join additional federations (giving their local userbase the opportunity to use these additional resources outside of the federation).

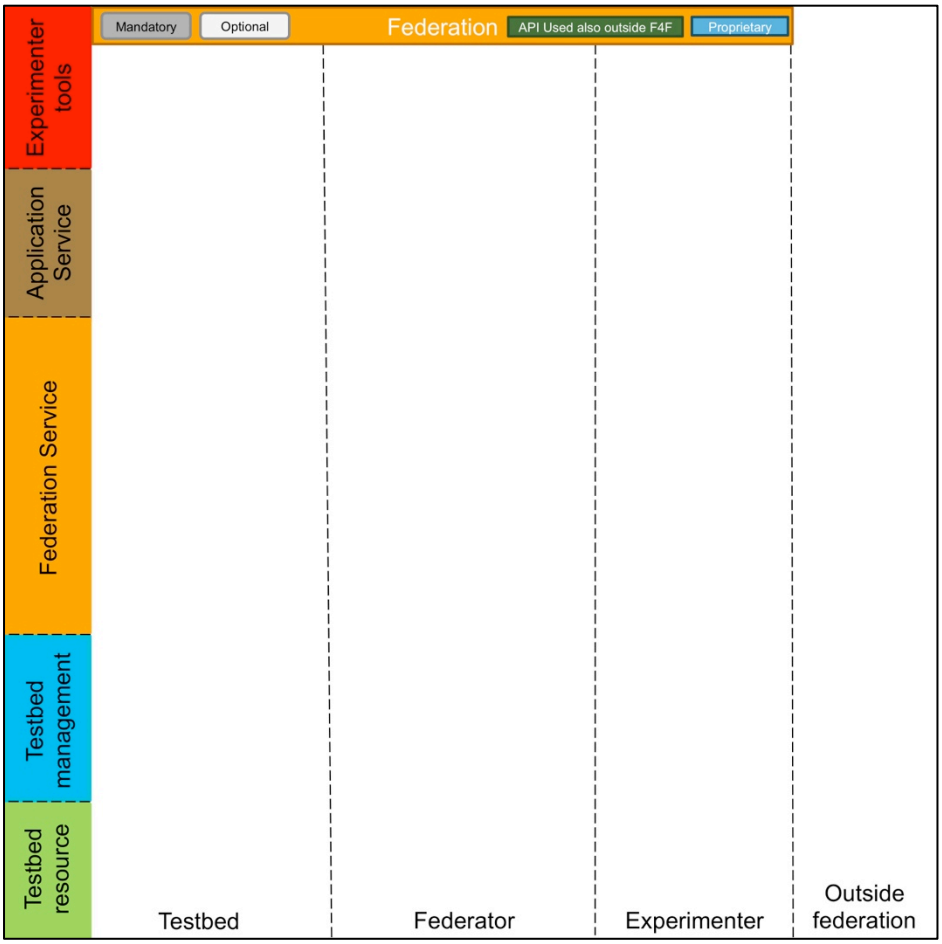


Figure 4: common format of the architectural diagrams

3.2.2 Layers in the architecture

In every architectural diagram, the same horizontal layers are defined (see Figure 4):

- **Testbed resources**: these are the physical and virtual nodes/resources of the testbeds
- **Testbed management**: this is the software framework which manages the resources of the testbed
- **Federation Services**: these are services that enable the federation to operate (e.g. authority services) or that make a federation easier to use (e.g. directory or broker services, documentation, portal, etc.). This is distinct from Application Services. Depending on what it actually is, a Federation Service may provide benefits to testbeds, experimenters, or both.
- **Application Services**: these are services that can be used by experimenters during their experiment (e.g. a service which returns sensor measurements in case of the SmartSantander testbed, or a service that can create a Hadoop cluster on nodes of the Virtual Wall automatically, instead of having the experimenter reserve Virtual Wall resources and then install such a cluster himself). Application services tend to abstract the underlying technical details of the provided services, and only provide direct benefit to an experimenter.
- **Experimenter tools**: this layer contains all tools the experimenters use

3.2.3 Colors in the architecture

Figure 5: Enlarged color legend of the architectural diagrams

As can be seen above, the architecture figures contain multiple colors:

- Grey (Mandatory): a component which is mandatory
- Dotted (Optional): a component which is optional
- Green: an API which on the server side is also used outside of Fed4FIRE or which is standardized. This means that an API can only be green if a service exists outside of the Fed4FIRE context that implemented exactly the same API.
- Blue: Proprietary API, or e.g. F4F API which means that there was an agreement in Fed4FIRE about this API interface.

3.3 Resource discovery, specification, reservation and provisioning

The first part of the architecture that is presented here aggregates all architectural components needed to support the following first steps of the experimental lifecycle: resource discovery, resource specification, resource reservation and resource provisioning. The architecture itself is depicted in Figure 6. The remainder of this section will first introduce all the corresponding components, and will then go into more details regarding the aspects for which additional information is required (SFA interfaces, mechanisms for authentication and authorization, the new layer for application services).

3.3.1 Introduction of the architectural components

3.3.1.1 Federator

From the perspective of the **Federator**, several components will be provided in one or more central locations for resource discovery, resource requirement, resource reservation and resource provisioning. You can really compare this to the Internet model:

- The basic thing you need is a server with a known IP address, and a client PC with a browser which connects to that server and shows the website.
- A DNS server can come in handy, as it translates e.g. www.cnn.com to the right IP address.
- On top of that, a search engine (e.g. google) is handy to find the right server if you are looking for information on Olympic Games e.g.

So the DNS server and search engine are not necessarily needed to use this internet, but they come in very conveniently.

The same is true for the Federator components. The only things which are needed to use a testbed are an experimenter tool, an authority (can be co-located with a testbed) and the testbed itself. However, the components below offered by the Federator make it more convenient, but they are not explicitly needed to let experimenters use the testbeds. This is also very important for sustainability aspects.

- **Portal**: a central starting place and registration place for new experimenters.
- **Member and slice authority**: experimenters who register at the portal are registered at this authority (as can be seen in Figure 6, there are also other authorities at testbeds). The APIs are not standardized.

- an **aggregate manager (AM) directory** which is readable by computers to have an overview of all testbeds available in the federation (more specifically, of their aggregate managers' contact information).
- a **documentation center** which gives an overview of available tools, testbeds and tips for the experimenter (http://doc.fed4fire.eu)
- **Authority directory**: for authentication/authorization between experimenters and testbeds, Fed4FIRE adopts a trust model where testbeds and authorities establish trust relationships among each other. To support authentication decisions at the testbeds, specific experimenter properties are included in the experimenter's certificate, which is signed by an authority. This approach allows testbeds to implement basic authentication functions or even rules-based authorization if they want to. To ease the creation of such a web of trust across the federation, there should be a trusted location that bundles all root certificates of the federation's authorities. This way a testbed can query/trust the central authority directory to see which root certificates it should trust.
- **Service directory**: directory service for federation and application services.
- **Future reservation broker**: a broker for resolving abstract reservation requests within the federation. Note that within WP5 the further details of this component will be specified. If that specification clearly indicates that the reservation broker is intended to be used as much for instant reservations as for future reservations, it might be more suitable to change the name of this element to just "Reservation broker". This remains to be seen.

From these components only the authority has state of experimenters and experiments, but per definition multiple authorities are supported and installed in the federation. The others are just directory services and documentation to help experimenters and tools find the right testbeds and the portal and future reservation broker, which make the life of an experimenter easy, but which are just some of the usable experimenter tools.

3.3.1.2 Testbed side

At the **testbed** side, we have the following components:

- A testbed provides **resources (=nodes).** These can be virtual or physical, and can be very diverse in terms of technology. In many (but not all) cases, these resources are reachable through SSH.
- A testbed management component (called the **aggregate manager**) that is responsible for the discovery, reservation and provisioning of the testbed's resources. The testbed has the freedom to adopt any desired software framework to implement this functionality, as long as it can expose these functions through the Aggregate Manager interface.
- A testbed may have an **authority** (member and slice) in the federation if it wants to. This makes the testbed independent of the availability of the Federator to allow its own experimenters to participate in the federation. If the testbed however considers this too much of a burden, and is confident that the Federator authority or another authority can provide a reliable and lasting service, then the testbed can choose to rely on the one of the other authorities of the federation to serve its own experimenters.
- The testbed may operate **application services** if it wants to. As will be explained in detail in section 3.2.2, these are services that can be used by experimenters during their experiment (e.g. a service which returns sensor measurements in case of the SmartSantander testbed). Application services tend to abstract the underlying technical details of the provided services, and only provide direct benefit to an experimenter. At the moment it is unclear which mechanism should be adopted for authentication and authorization by application services inside the federation. From the WP2 Architecture point of view it seems preferable

that all Fed4FIRE federation services would rely on X.509 certificates for this, since this would allow the experimenters to use the same user certificate for the application services as they already do today for the testbed resources. However, before making any final recommendations, it is required that WP7 performs a more thorough feasibility check and technical specification of this aspect.

3.3.1.3 Experimenter side

From the **experimenter** point of view, we can distinguish the following components:

- Some of the tools made available to the experimenter are hosted tools (e.g. the portal, future reservation broker, documentation center, possibly some application services, etc.). The experimenter will make use of these tools through a **browser**.
- Several experimenter tools for resource discovery, reservation and provisioning already exist and run locally on the experimenter's computer. So these are **stand-alone tools** instead of hosted tools. Examples are Omni, SFI, NEPI and jFed. The experimenter has the freedom to choose any tool of his or her preference, it will be supported by Fed4FIRE as long as it is compatible with the adopted AM APIs.

3.3.1.4 Outside of the federation

The last actor depicted in Figure 6 relates to the entities **outside of the federation.** The following components are relevant:

- Just as the testbeds in our federation, the testbeds outside of the federation provide **resources**.
- A **testbed manager** of some sort manages these resources. It can expose its functionality through any desired API. If it supports the AM API, then it is compatible with all Fed4FIRE tools. If it wants, this testbed can decide to authorize (specific) Fed4FIRE members, even without formally belonging to the federation.
- Any party outside of the federation can provide **application services** to both Fed4FIRE and non-Fed4FIRE members. These application services can even rely on Fed4FIRE resources if the external application provider has approval of the specific involved Fed4FIRE testbeds for doing so.
- These application services can expose themselves using any interface that they want. Therefore they can have their own credentials or login/password for authentication and authorization of their users. The **service authority** is then responsible for the management of this service-specific identity information.

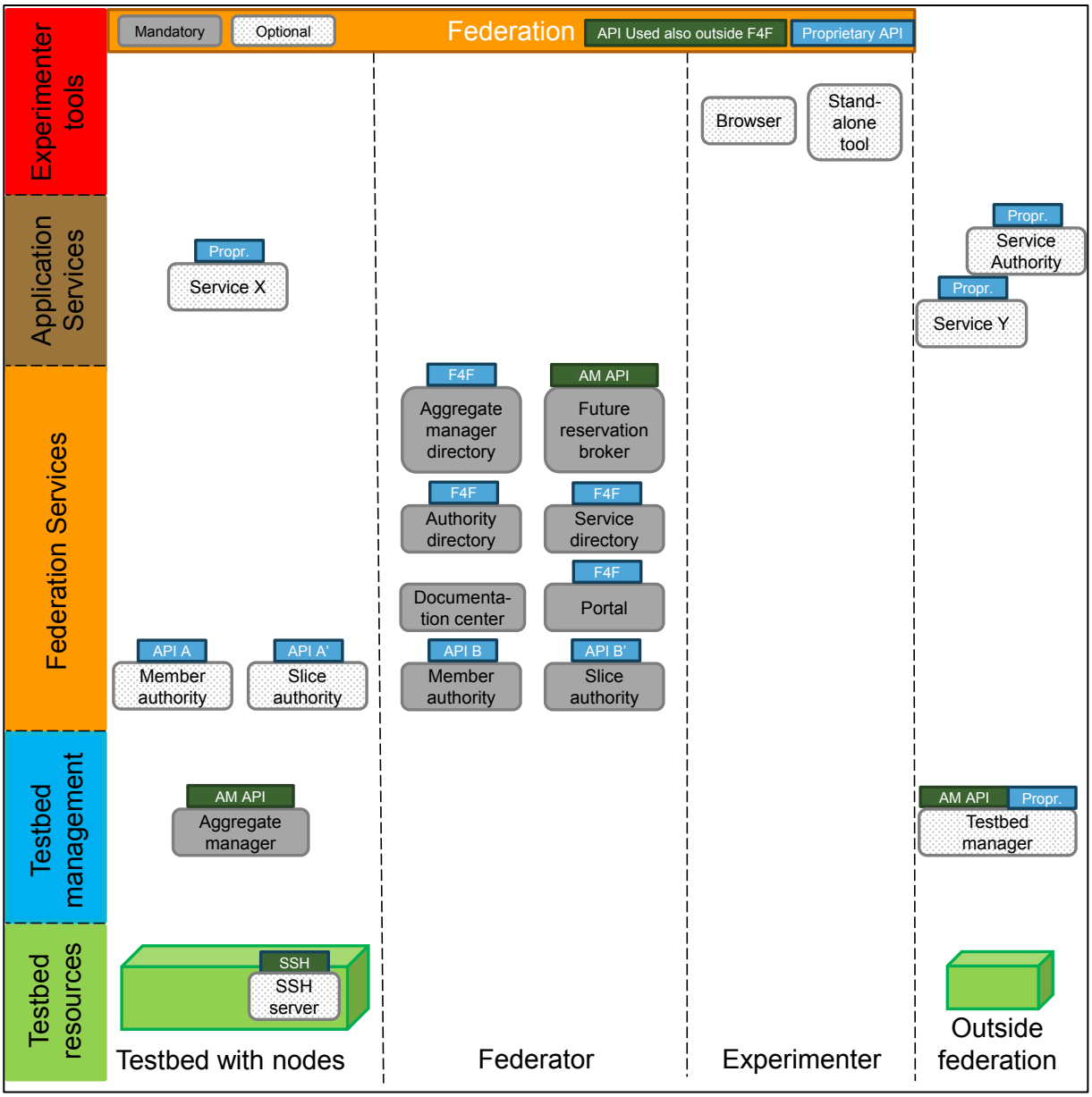


Figure 6: Fed4FIRE cycle 2 architecture for discovery, reservation and provisioning

3.3.2 Basic concepts

Several aspects of the architecture presented in Figure 6 originate from the Slice-based Federation Architecture (SFA) [9]: the Aggregate Manager API, the member authorities and the slice authorities. Given the importance of these components in the Fed4FIRE architecture, some more detail is given in this section.

3.3.2.1 Concepts slice, sliver and RSpec

Three key concepts are slices, slivers and RSpecs. As depicted in Figure 7, a **slice** is the concept that is used to bundle resources belonging together in an experiment or a series of similar experiments, over multiple testbeds.

A **sliver** is the part of that slice which contains resources of a single testbed. One uses an **RSpec** (Resource Specification) on a single testbed to define the sliver on the testbed. The RSpec and thus the sliver can contain multiple resources. Note that there is an important difference between the concept sliver as defined in the Aggregate Manager API v2 and v3.

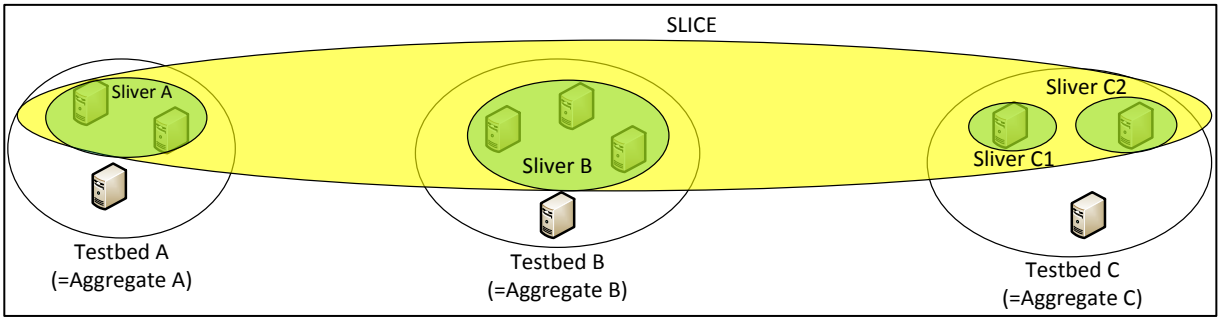


Figure 7: Concepts of slice and sliver

Version 2 of the AM API supported only one sliver per slice per testbed [6]. Hence in this version of the AM API a sliver denotes all the slice's resources on a testbed. (e.g. the deletesliver call deletes all resources of a slice on a single testbed). In the figure above Sliver C1 and C2 cannot exist in AM API v2 (of course, a testbed can decide to work internally as such, but the slivers are not individually addressable through the API).

In version 3 of the AM API, one of the several important problems that was solved is that of adding resources of a testbed to a slice. In AM APIv2, as only a single sliver is supported per testbed and slice, it was not possible to add extra resources to this same slice for that testbed (and remove them afterwards). For this, API v3 now supports addressing of individual slivers, and the SFA calls such as allocate, provision, etc. can now work on multiple slivers on a single testbed. For instance the call "allocate" returns a struct of slivers.

In practice it is a little more complicated since testbeds have the total freedom to adopt one of multiple methods when allocating resources to slivers [8]. If you request an RSpec with multiple resources, a testbed can decide to create multiple slivers, only a single one, or any allocation strategy in between. This has then as a consequence that it depends on the testbed if an experimenter can remove single resources/slivers or only everything at once.

3.3.2.2 GENI AM API and GENI RSpecs

The current concepts and APIs of the GENI framework are documented on the GENI wiki [10]. The key components and interfaces are the following:

- GENI Aggregate Manager (AM) API version 3 [7]
  - This contains a description of the API for discovery, resource requirements and provisioning.
  - It defines the GENI certificates for authentication. The GENI AM API uses XML-RPC over SSL with client authentication using X.509v3 certificates.
  - It defines GENI credentials for authorization
  - It defines GENI URN identifiers for identifying and naming users, slices, slivers, nodes, aggregates, and others
- GENI RSpec (resource specification) version 3 [11]. The RSpec comes in three flavors:
  - Advertisement RSpec: for resource discovery (getting a list of all resources)

- Request RSpec: requesting specific resources
- Manifest RSpec: describing the resources in an experiment

The services that we identified as 'federation services' are in GENI partly identified as a Clearinghouse [12]:

- An Identity Provider (IdP) provides certificates and PKI key materials to human users, registering them with the GENI federation as GENI users.
- A Project Authority asserts the existence of projects and the roles of members (e.g. PI, Experimenter).
- A Slice Authority provides experimenters with slice credentials by which to invoke AM (Aggregate Manger) API calls on federation aggregates.
- A Service Registry provides experimenters with a 'yellow pages' of URL's of all trusted services of different kinds. In particular, the list of all available aggregate managers trusted by GENI (possibly satisfying particular criteria) is provided.
- A Single-Sign-on Portal, which provides web-based authentication and access to the authorized Clearinghouse services and other GENI user tools.

Fed4FIRE uses as such the GENI AMv3 API, and is working together with GENI on a Common AM API as a next version.

3.3.3 Details of the adopted mechanisms for authentication and authorization

One of the advantages of the GENI AM API is that authentication and authorization is based on X.509v3 certificates, credentials and a chain of trust. This means that by nature it is distributed and very scalable. Since Fed4FIRE adopts the GENI AM API, it inherently adopts the same mechanisms for authentication and authorization.

More specific, the following authorities will be trusted by the testbeds in cycle 2 of Fed4FIRE:

- Fed4FIRE central member and slice authority (through http://portal.fed4fire.eu)
- PlanetLab Europe member and slice authority (through http://www.planet-lab.eu/)
- Virtual wall member and slice authority (through http://www.wall2.ilabt.iminds.be)

Currently the APIs on top of these authorities are proprietary.

3.3.4 Details of the layer "Application Services"

Through the practical experience with the development of the first cycle of the project, and through the requirements collected in the scope of WP4 (reported in D4.2 - Second input from community to architecture [3]), the need was identified to introduce a new layer in the architecture that is intended to make it easier for experimenters to use the resources of the federated infrastructure in their experiments. This is the layer called "Application Services" in Figure 6. Some examples might clarify what is exactly meant here.

For instance, in the case of SmartSantander you could have the experimenter reserve the real sensing devices, and collect the data during the course of the experiment. However, SmartSantander also deploys an application service that already gathers all this data or processed versions of them, e.g. historical, maximum, minimum, etc.) , and exposes it through an API that can be called by the experimenters.

Another example is that of a Hadoop cluster (big-data framework) on resources of the Virtual Wall. If an experimenter wants to setup such a cluster, he can reserve some nodes on the Virtual Wall, and install a Hadoop cluster on them manually. However, if this is something that many people seem to be doing, and if such an installation is quite labor-intensive, it makes sense for the Virtual Wall administrators or experimenters or application providers to create an application service that can create such Hadoop clusters automatically. In fact, the experimenters shouldn't even be aware of the fact that this cluster is being deployed on the Virtual Wall, from their perspective it should be sufficient to know that they asked for a Hadoop cluster with certain characteristics to be deployed, and that this was actually setup for them.

The introduction of such a new layer brings some important questions that need to be answered in the design of the federation architecture. One issue to solve is the fact that it is necessary that service providers (federated parties or 3rd parties also as external service providers) can publish their services towards the experimenter community. Another question is the definition of a suitable interface to expose the actual application service to the experimenters. A third unknown factor is how to support the different levels of abstraction of the underlying technology in various conditions. These questions are tackled in the remainder of this section.

3.3.4.1 How to publish application services?

As mentioned above, the services have to be published in order to allow the experimenters to discover them and to decide if they fulfill their needs. A service directory that gathers all the available services and their descriptions is added to the architecture for this. The exact API for this and implementation details will be worked out in WP5. It should contain a link to the documentation of the service where one can find the used protocol, the method to authenticate and the exact API details.

3.3.4.2 How to expose application services?

Services provide an added value, so the most important aspect of services is the logic they implement (Experiment Service Logic), and the interfaces should only be a means to expose these services. Whereas interfaces might evolve over time, the service logic remains. This allows operating independently of a specific technology and allows more possibilities for service invocation (for example, an orchestrated workflow could be implemented in the future).

In terms of protocols for the application services, we go along with what is suitable for a specific service. We consider different protocols to provide services: HTTP-REST, SOAP, RPC-XML and others. The documentation of the protocol and API (see 3.3.4.1) is key for using it correctly.

3.3.4.3 High level examples of application services

The service provider exposes the added-value services, deals directly with the experimenter and interacts with the different resources, tools and utilities within each testbed in order to deliver what the experimenter requests. The experimenter might be mainly interested in the functionality the service provides and not so much in the underlying details required to implement it. This allows the experimenter with limited technical skills -or simply not interested in technical details- not to interact directly with the testbeds but rather focus on the service delivery. In this case, the experimenter is not aware or does not care about the different testbeds involved in his experiment (and their

internal resources, tools and utilities); it is the service provider who manages all the interactions with the different testbeds.

Examples of this are for example a service provider which offers a service to deploy Hadoop clusters (where the Hadoop user does not know which underlying physical testbeds and resources are used) or a service to read out sensor values of a Smart City (e.g. SmartSantander, where the experimenter is not involved with the real sensors).

Hence, the service provider deals with the whole experiment lifecycle, as a service to the experimenter -discovering testbeds, negotiating reservations and access to the testbeds, uploading input data, running and monitoring the experiment, downloading any output data and presenting the results to the experimenter. The service provider is the single contact point for the experimenter. In this scenario, the experimenter does not need to be identified to the federated environment in order to access the different testbeds, since it is the responsibility of the service provider to authorize and authenticate the user who wants to use the exposed service. It is the service provider who has SLAs and agreements with the testbeds offering physical resources.

It will be worked out further in WP5, but it might be also possible to use the same X.509 credentials as for the AM API, to speak to the service, and then it is also possible that the service speaks to the testbeds in name of the experimenter himself and that the service automates everything, but that the SLAs are between the experimenter and the testbeds.

3.3.5 Sequence diagrams for resource discovery, specification and provisioning

To further clarify the role of all the architectural components needed for discovery reservation and provisioning, and to illustrate how they interact with one another, the corresponding sequence diagrams are given below. Figure 8 describes the very first steps that a new experimenter would take in the Fed4FIRE federation. It all starts when the experimenter (we will assume that this is a man in the remainder of this discussion to keep the text readable) has a specific idea, and wonders if he could experiment with it on Fed4FIRE (1). To assess if this is actually the case, he visits the project's documentation center (which is completely open to everyone) and browses the web-based catalogue of Fed4FIRE testbeds (2). Once the experimenter has identified the testbeds that he wants to experiment on, he surfs to the First Level Support dashboard (which is also completely open to everyone) to check if the testbeds are up and running and if they have free resources (3). As a last step of his exploration of Fed4FIRE, he browses to the service directory to look for application services that could minimize the needed development for experimenting with his idea (4). Note that the documentation center, the First Level Support dashboard and the service directory are three different architectural components, but they can (and should) be implemented with cross links.

Since the outcome of all previous steps was positive, the experimenter decides to actually experiment with his idea on Fed4FIRE testbeds (5). The next step therefore will be to get an actual Fed4FIRE user account. For this he retrieves the documentation regarding the registration process from the project's documentation center (6). Based on this guidance documents, he is able to register for an account on one of the Fed4FIRE member authorities (7). Once the administrator of that authority has approved his request, he goes back to that authority and retrieves his X.509-compatbile Fed4FIRE member certificate (8). Now that he has a valid certificate, he only has to decide which tools to use with that certificate to start discovering, reserving and provisioning resources. For this, he surfs again through the documentation center to browse through the catalogue of different experimenter tools that he could use (9). After deciding which specific tool(s)

he wants to use for his experiment, the experimenter is entirely ready to start experimenting on Fed4FIRE (10).

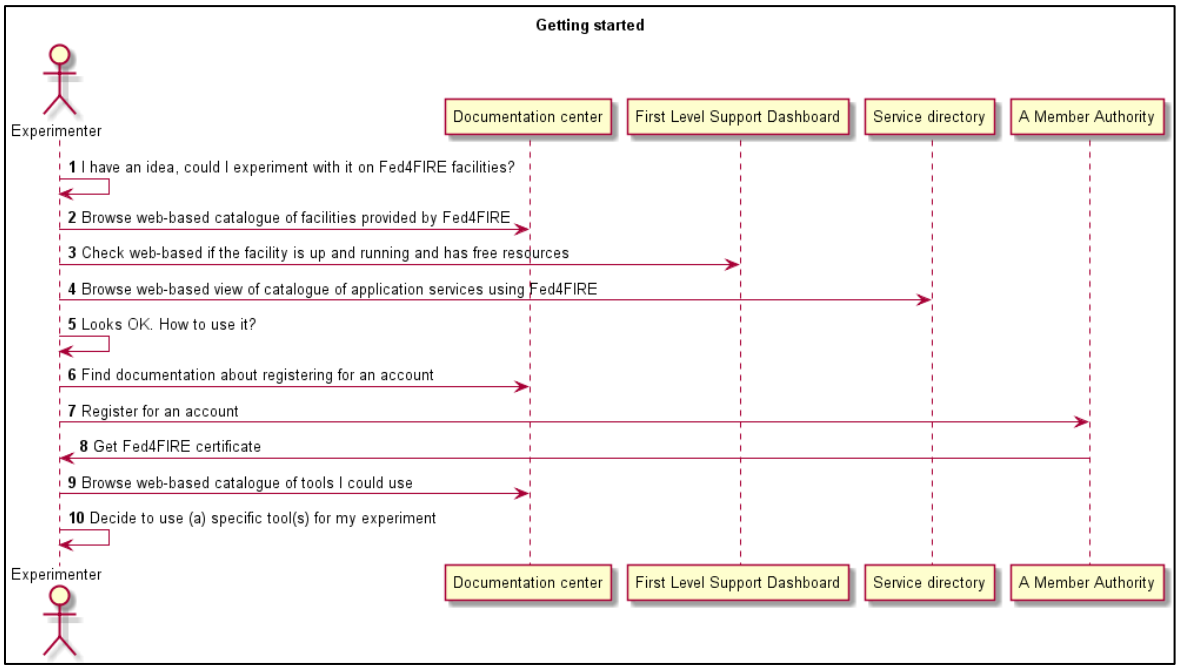


Figure 8: Sequence diagram for initial experiment lifecycle

The next steps to actually setup the experiment are shown in Figure 9. This sequence diagram shows the provisioning right now, without advanced reservation. The first thing that the experimenter has to do is to provide his Fed4FIRE member certificate (which he downloaded before) to the experimenter tool(s) that he wants to use (1). In case of a hosted tool this can be a derivative certificate (e.g. a delegated credential, or a speaks-for credential). Once he has done this, the tool is ready to be used by him. The first thing the tool will want to do is to create a slice for the experiment. The first step for doing so is to learn the contact information of the corresponding member and slice authorities. The tool will retrieve this information from the authority directory (2, 3). The next step will then be to retrieve his user credential from the member authority (4, 5). This credential provides information regarding the specific rights that were given to him by the member authority. One of them is the right to create slices. Using this credential, the experimenter can then register a new slice at the slice authority (6, 7). When creating a slice, an expiration date (end date) has to be given. This can be later made longer (but not shorter).

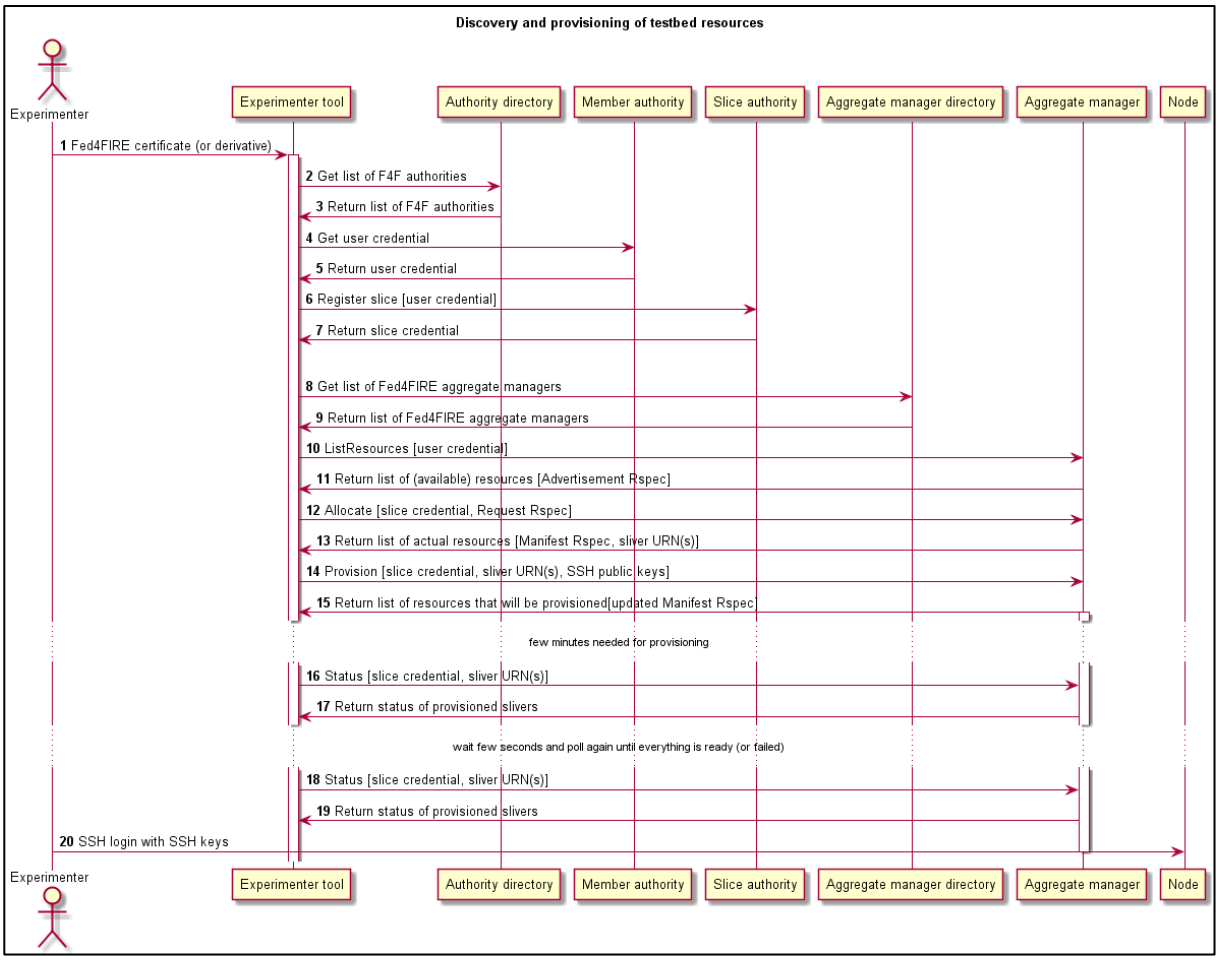


Figure 9: Sequence diagram for discovery and provisioning

Now that the experimenter is the owner of his slice, he can start adding resources to it. For doing this, his tool will first need to retrieve the contact information for the different testbeds (more specifically the aggregate managers) from the aggregate manager directory (8, 9). This information allows the tool to ask for a list of (available) resources at every testbed of the federation (10). This information is returned by the testbed in the form of an Advertisement RSpec (11). Once the tool has presented an overview of all this information to the experimenter, and he has finally selected the specific resources he wants to use, the next step will be to actually request these resources at the corresponding testbeds. For this the tool will create the appropriate Request RSpec, and will request the testbed's aggregate manager to allocate the corresponding resources to the slice of the experimenter (12). To prove that the experimenter has the right to add resources to this slice, he will attach the slice credential that he received from the slice authority in step (7). The aggregate manager will then decide if he can or cannot assign these resources to that slice, and will communicate the corresponding result to the experimenter tool in the form of a Manifest RSpec (13). He also returns the URN(s) of the different slivers if the assignment was successful. However, even in case of a positive response, being allocated does not yet mean that a resource is usable by the experimenter. It first needs to be provisioned. For this the experimenter tool will request the testbed's aggregate manager to provision the resources (14). It will attach the slice credential, sliver URN(s) and the public SSH key of the experimenter to this request. The aggregate manager will respond with an updated Manifest RSpec, describing the list of resources that will be provisioned (15). After waiting a few minutes while the aggregate manger performs the actual provisioning of the resources, the experimenter is eager to start using the resources. For this he will start checking the status of the requested resources (16, 17). Since the resources were not yet ready, he waits a bit

longer, and asks for the status again (18, 19). This time the response indicates that the resources are ready to be used. The experimenter then logs in with SSH on the resources (20) using a key pair of which the public key was attached as an argument to the provision request in step (14).

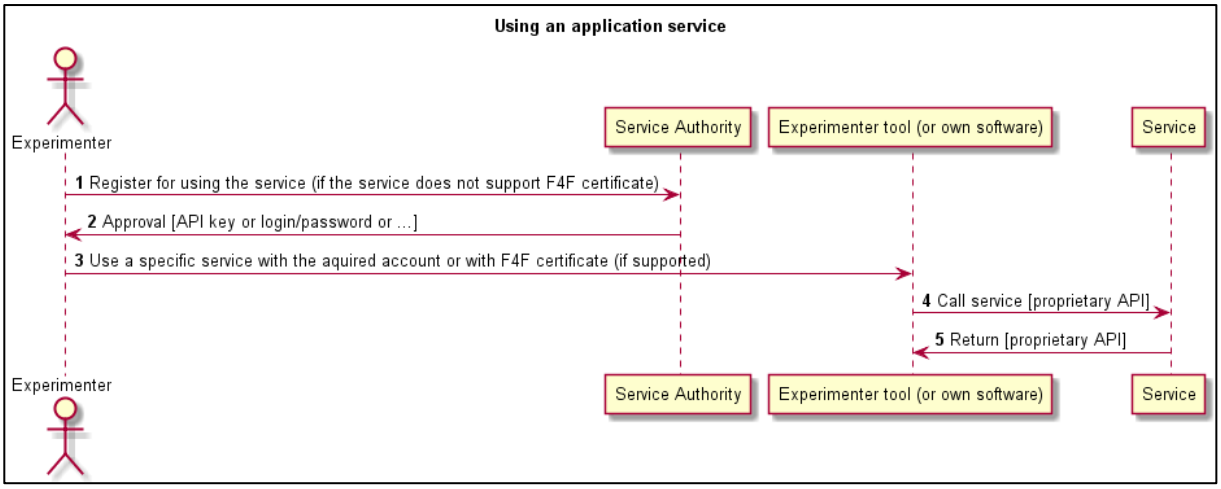


Figure 10: Sequence diagram for using an application service

The last aspect of this part of the federation architecture that needs some more explanation through a sequence diagram is the usage of an application service. As described in section 3.3.1, it is allowed that the service may adopt its own authentication and authorization methods. Inside the federation it is possible that later on it will be obliged to support the Fed4FIRE X.509 certificates.

As depicted above in Figure 10, the first step when using an application service is to register for that service at the corresponding Service Authority (1, 2). Once the needed credentials have been retrieved for the application service (being the experimenter's X.509 Fed4FIRE member certificate or any other credential specific to the application at hand), the experimenter can hand these over to its tool that will make use of the service, together with a request to use a specific service (3). The tool will then call the service (4), and retrieve the results (5).

3.3.6 Resource reservation (in the future or instantly)

With resource reservation we mean the reservation of resources in the future (e.g. next week Tuesday and Wednesday) or instantly (starting right now). Especially testbeds offering physical resources do need this if their occupation is high or if you need very specific resources (e.g. for wireless experiments). This is planned to be implemented for cycle 2 on at least four testbeds (Nitos, NETMODE, PLE, Virtual Wall and w-iLab.t) and probably also on Bonfire.

Currently, the prototypes are being implemented but it seems too early to put definitive API call sequences in this deliverable as it seems that implementation may still vary too much. We will adopt this in the architecture deliverable of cycle 3 based on the real implementations that will be available and verified by then.

It should be noted that the instant reservation as described in the previous sections is just a special case of this with the starting time 'now' instead of in the future. Slice and sliver creation have expiration/end times, so it is a special reservation in the future having as a starting point 'now' till the 'expiration date'.

3.4 Monitoring and measurement

The second part of the architecture that is presented here aggregates all architectural components needed to support monitoring and measurement. The architecture itself is depicted in Figure 11. The remainder of this section will first introduce all the corresponding components, and will then go into more details regarding the specific architecture of the monitoring for the First Level Support function.

3.4.1 Introduction of the architectural components

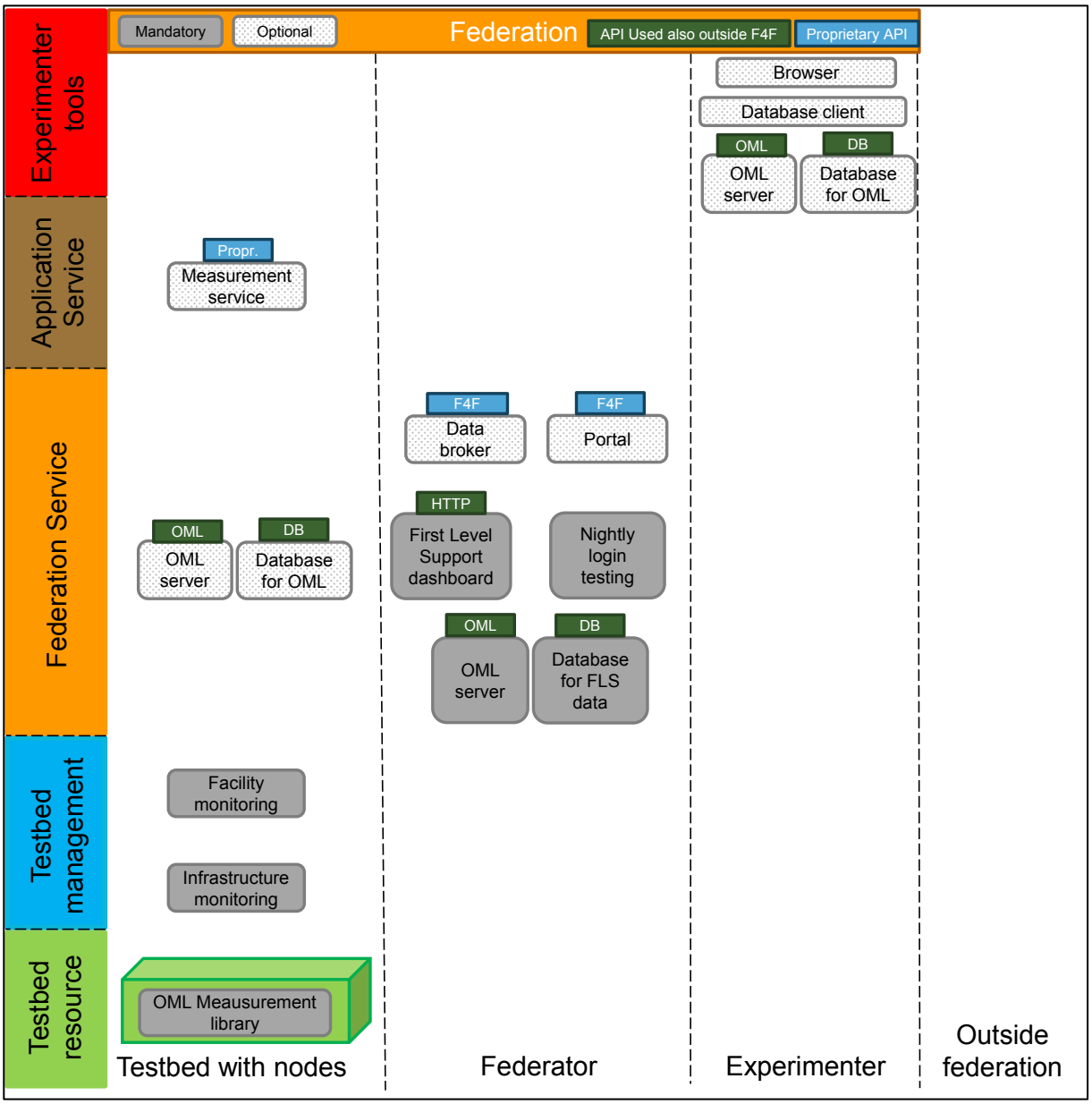


Figure 11: Monitoring and measurement architecture for cycle 2

At the **testbed** side, the following architectural components can be distinguished:

- **Facility monitoring**: this monitoring is used in the first level support to see if the testbeds are up and running. The testbed has the freedom to adopt any solution to gather this type of monitoring data as it sees fit (e.g. an existing monitoring framework such as Zabbix, Nagios or similar), as long as it is able to export that data as an OML stream to the Federator's central OML server, which will store it in a database for First Level Support. In the first cycle of Fed4FIRE, the facility monitoring was rolled out on all testbeds. More details are given in section 3.4.2.
- **Infrastructure monitoring:** Instrumentation of resources by the testbed provider itself to collect data on the behavior and performance of services, technologies, and protocols. This allows the experimenter to obtain monitoring information about the used resources that he could not collect himself. Examples of such infrastructure monitoring data are information regarding the CPU load and NIC congestion on the physical host of a virtual machine resource, the monitoring data of switch traffic, or the gathering of data regarding the wireless spectrum during the course of the experiment.
- **OML measurement library** for experiment measuring**:** this component is intended for measurements which are done by a framework that the experimenter uses and which can be deployed by the experimenter itself on his testbed resources in his experiment. The experimenter can retrieve or calculate this data as he prefers, and can then use the OML measurement library (which should be available on every resource) to easily export it to an OML server with database for storage.
- **OML server**: this component can be configured to be the endpoint of a monitoring or measurement OML stream, and can store this data in several types of **databases** (PostgresQL, SQLite3). The deployment of an OML server by a testbed provider is optional. The best way to retrieve the data should be further investigated in WP6. A common practice today is to get the data directly from the database using the raw database API. This is most likely beneficial in terms of performance, but on the other hand this means that the API for data retrieval differs per underlying database technology. An alternative would be to create an OML-focused API that might be a little less optimal in terms of performance but that can be consistently implemented over a range of database products, and which is in line with the decision to adopt OML as the common interface for measurement and monitoring in the project. A more detailed analysis of the pros and cons of both approaches is needed, together with a feasibility assessment of both approaches.
- The testbed provider that provided an OML server can decide to make this data easy to retrieve for the experimenter by exposing it as a **measurement service** using a proprietary interface. The deployment of such a measurement service is optional.

When focusing on the **Federator**, the following elements of Figure 11 require some further introduction:

- The **FLS dashboard** gives a real-time, comprehensive but also very compact overview of the health status of the different testbeds included in the Fed4FIRE federation. To determine this health status it combines facility monitoring information provided by the testbeds with specific measurements performed by the dashboard component itself. More details about this are given in section 3.4.2. The existence of the FLS dashboard at the Federator level is in line with the project's approach that Federator components should be put in place for convenience, but shouldn't be critical for the federation's operation. The dashboard is indeed a very useful tool to have, but if its operation were disrupted or even discontinued, experimenters would still be just as able to get resources and work on them as when the dashboard was still supported. Also, since it is a stateless component (it combines data from the different testbeds in a single health overview), it can easily be duplicated or moved. This

would only result in a loss of historical information about the federation's health, but no functionality loss would occur.

- The federator provides an **OML server and corresponding database for FLS data** to process and store facility monitoring data of the testbeds to be used by the First Level Support (FLS). These two Federator components (which are depicted in Figure 11) are a necessity to implement the FLS dashboard, but since the dashboard itself is not critical to the federation, neither are the central OML server and corresponding database. This motivates that the Federation level is a suitable place for these components.
- The component for **nightly login testing** provides a second view on the operational status of the federation. Its information is not real-time (typically tests would automatically be performed once or twice a day), but the result of these tests is more thorough then those of the FLS dashboard. This is because in this case the testing module performs an actual experiment (including all steps of the experimental lifecycle related to resource discovery, reservation and provisioning, ending with an actual automatic SSH login on the provisioned resources) and tracks success or issues for any intermediary step of the experiment lifecycle. According to exactly the same principles as for the FLS dashboard, the component for nightly login testing is in line with the approach that Federator components should not be critical for the operation of the federation (disruption in the nightly login testing does not hinder the execution of experiments, this testing could be easily duplicated or moved).
- The **data broker** is an optional component that can be **accessed through the portal**, and which makes it easier for novice experimenters to retrieve their experiment data from the different sources where they might reside (OML servers of the different testbeds that provided infrastructure monitoring, OML servers of the experimenter itself on which the experiment measurements were stored, etc.). According to exactly the same principles as for the FLS dashboard and the component for nightly login testing, the data broker is in line with the approach that Federator components should not be critical for the operation of the federation (disruption in the data broker service does not hinder the direct retrieval of experiment data, the broker service could be easily duplicated or moved).

The **experimenter** himself can also utilize different experimenter tools:

- His **browser** is used to access the data broker through the portal.
- A **database client** allows the experimenter to retrieve his monitoring and measuring data from any OML server where it was stored
- For storing infrastructure and measurement data, the experimenter has the option to deploy his own **OML server and attached database** in order to archive the data himself.

3.4.2 Sequence diagrams regarding measuring and monitoring

The different steps involved in **facility monitoring** are depicted in Figure 12. In short, a monitoring agent on the selected key component(s) of the testbed provides its data to the testbed's facility monitoring component (1). The testbed has the freedom to adopt any facility monitoring framework as it sees fit (Zabbix, Nagios, proprietary software, etc.). This component will calculate the overall testbed health status (2), and will push this health status info to the central OML database accordingly (3). This makes it straightforward for testbeds to support facility monitoring, since the logic is kept locally, and the mechanism to export the result to the central component is OML, which is the same as that for the infrastructure monitoring and experiment measurement functionalities. Testbeds could also easily support multiple central OML servers by just exporting different copies of this OML stream to the different central servers.

In the next step, an experimenter is then able to browse to the FLS dashboard to look at the status overview of the Fed4FIRE testbeds (4, 5). For depicting this overview, the dashboard first has to collect the corresponding data from the central OML database (6, 7), after which it can display the status of the testbeds to the experimenter (8, 9).

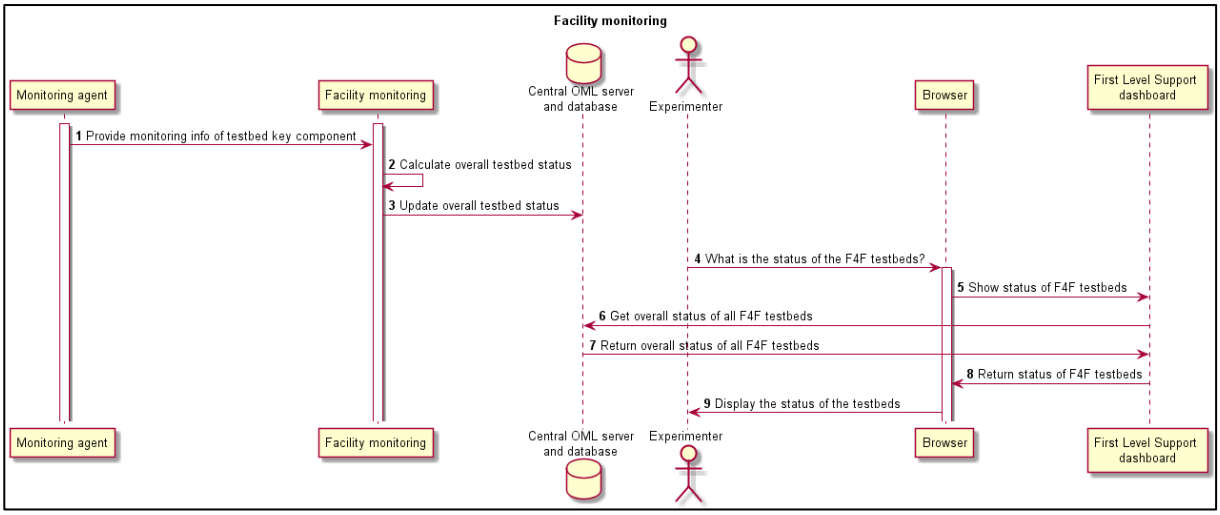


Figure 12: Sequence diagram for facility monitoring

The steps related to **infrastructure monitoring** are summarized in Figure 13. The first step is to collect the data using monitoring agents deployed on the resources. These agents export their data to the infrastructure monitoring framework (1, 2), which will then have to expose it through an OML interface. When requesting resources, the experimenter has to provide information of where the monitoring data should be exported to (exact location of central or experimenter OML server). This then allows the experimenter to use its database client to retrieve his infrastructure monitoring data from the corresponding OML server and database (3-6). In the case the infrastructure monitoring data is spread across different OML servers, the experimenter has the option to easily retrieve all this data through the Fed4FIRE data broker (7-13).

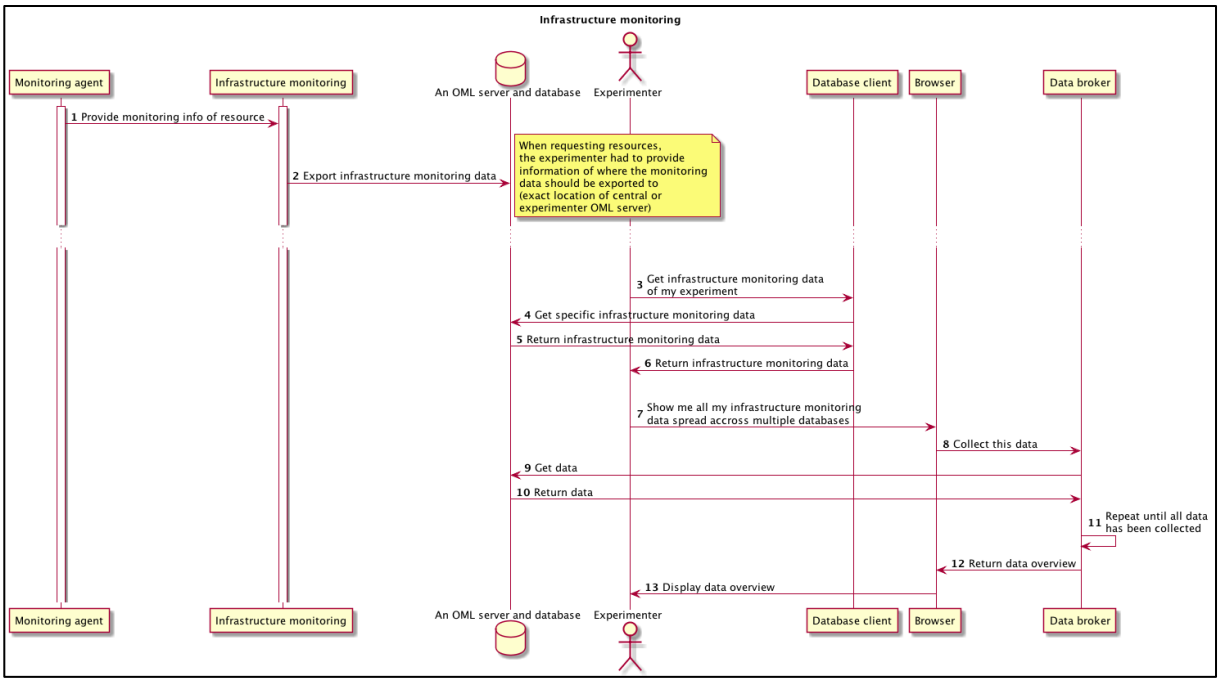


Figure 13: Sequence diagram for infrastructure monitoring

The sequence diagram regarding **experiment measuring** is depicted in Figure 14. In short, it is practically identical to that of infrastructure monitoring that was given in Figure 13. The only important difference is that in this case the data source is any piece of experimenter software that makes use of the measurement library. This library stores the experiment measurements directly in the OML server which was appointed by the experimenter (1). All other steps for retrieving this data as an experimenter are the same is for infrastructure monitoring (2-12).

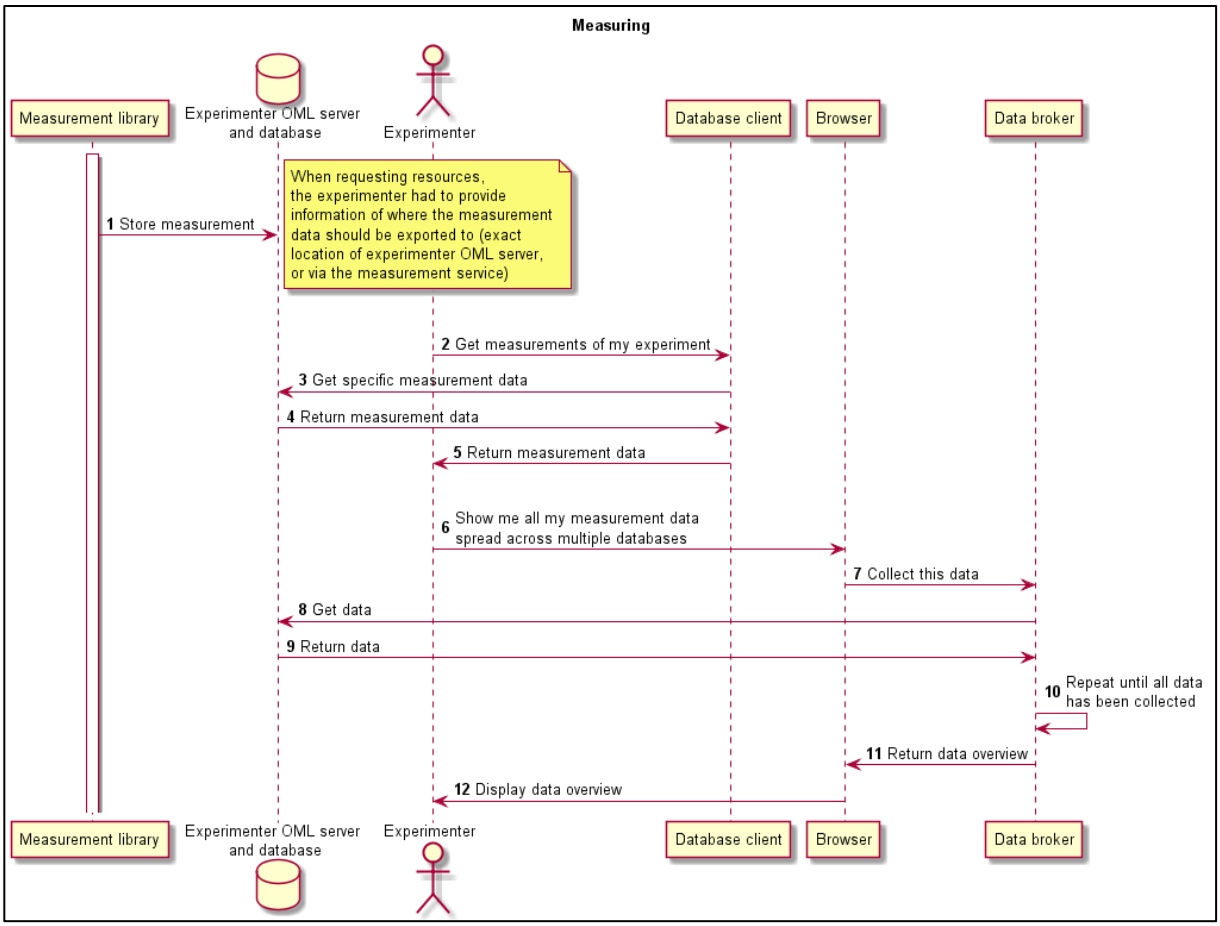


Figure 14: Sequence diagram for experiment measuring

3.4.3 Monitoring and measuring for First Level Support

One of the aspects of this part of the Fed4FIRE architecture that deserves some more details is the specific adopted approach for monitoring and measuring for First Level Support. Currently the following is deployed as depicted in Figure 15. We monitor 5 things per testbed:

1. ICMP ping to the AM server or some other testbed server: this checks connectivity over the internet to the testbed (if this fails, testbed can likely not be used)
2. AM API GetVersion call: tests the AM component (no credential is needed)
3. AM API Listresources to know the number of free resources (if this is 0, then new experiments cannot be created)
4. Red/Green/Amber aggregation state of what the testbed provider monitors himself (this is custom per testbed, and based on the testbed's facility monitoring data)
5. Last timestamp of the information monitored at the testbed itself and injected into the OML server: if this timestamp is too old, the testbed monitor doesn't provide any relevant information anymore.

There is a live visualization at https://flsmonitor.fed4fire.eu and there are also email alarms and long term statistics to dive into the monitoring information of the past.

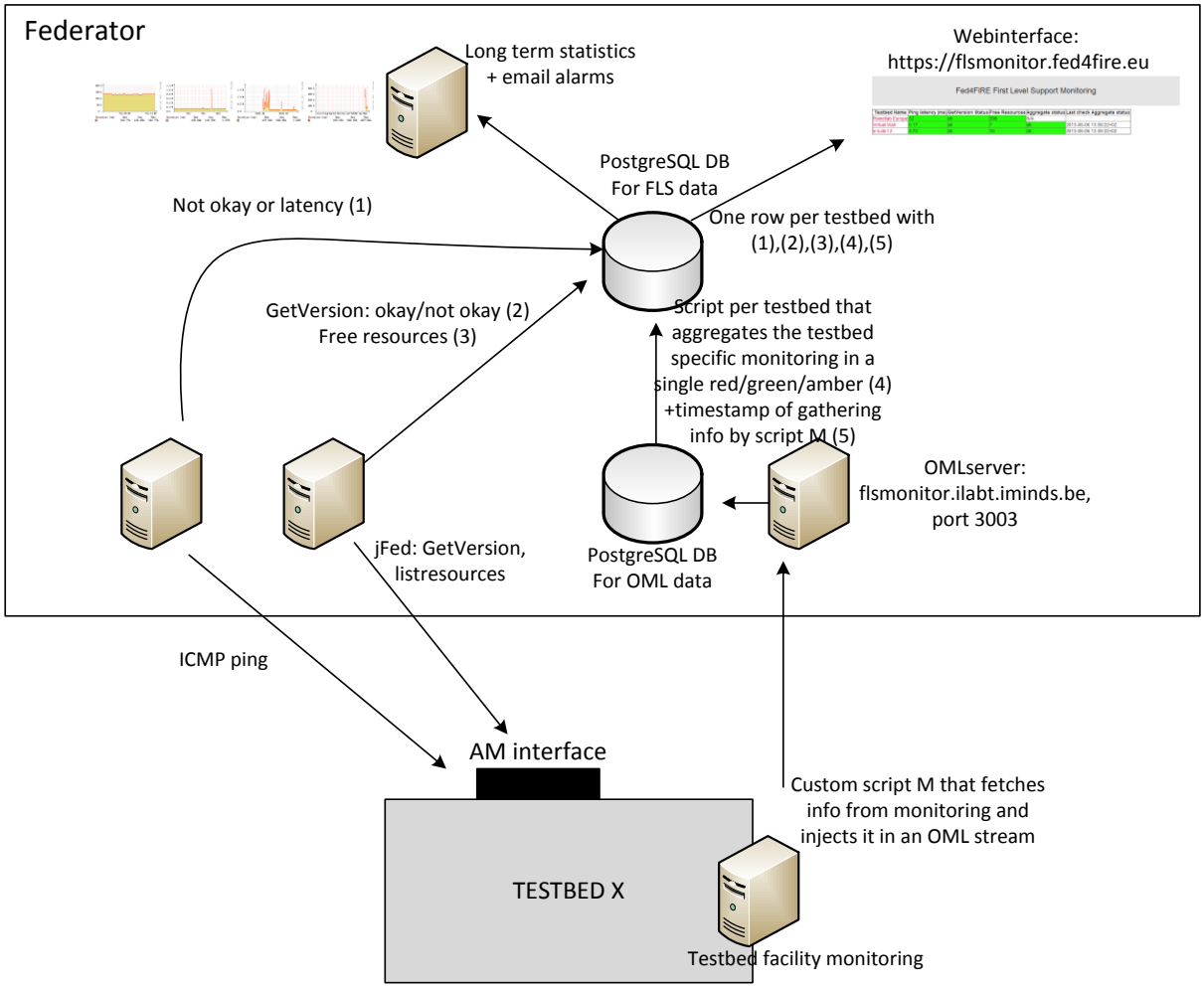


Figure 15: FLS dashboard architecture

Since a picture says more than a thousand words, a screenshot from the FLS dashboard (taken while the dashboard was still under development) is given below.

Fed4FIRE First Level Support Monitoring

Testbed Name	Ping latency (ms)	GetVersion Status	Free Resources	Internal testbed monitoring status	Last check internal status
BonFIRE	31.17	N/A	N/A	ok	2014-02-21 09:45:59+01
Fuseco	35.77	ok	1	ok	2014-02-21 09:45:03+01
Koren	284.47	N/A	N/A	N/A	N/A
NETMODE	64.95	ok	20	ok	2014-02-21 09:40:22+01
NITOS Broker	66.22	ok	38	ok	2014-02-21 09:45:02+01
NITOS SFAWrap	30.59	ok	60	N/A	N/A
Norbit	N/A	N/A	N/A	ok	2014-02-21 09:36:32+01
Ofelia (Bristol island)	15.39	N/A	N/A	ok	2014-02-21 09:45:02+01
Ofelia (i2CAT island)	N/A	N/A	N/A	ok	2014-02-21 09:45:02+01
Planetlab Europe	30.78	ok	286	ok	2014-02-21 09:45:02+01
SmartSantander	85.3	ok	0	ok	2014-02-21 09:40:01+01
Virtual Wall	0.22	ok	7	ok	2014-02-21 09:44:39+01
w-iLab.t 2	6.48	ok	23	ok	2014-02-21 09:44:58+01

Figure 16: FLS dashboard screenshot

3.5 Experiment control

The third part of the architecture that is presented here aggregates all architectural components needed to support experiment control. The architecture itself is depicted in Figure 17. The remainder of this section will first introduce all the corresponding components, and will then further clarify them through the appropriate sequence diagrams.

3.5.1 Introduction of the architectural components

At the **testbed** side, the following architectural components can be distinguished in Figure 17:

- The simplest mechanism for experiment control is allowing the experimenter to login to the nodes using SSH, and to control the experiment manually. In this case it is required that an **SSH server** is deployed on the resource.
- The **resource controller** is an agent that runs on the testbed's resource, and which can invoke actions on the request of the experimenter on that resource. These actions have to be communicated to the resource controller in the FRCP protocol. Optionally a resource controller can also invoke actions on another resource which is connected through a communication link. An example is a resource controller running on a node and controlling a sensor through USB.
- These FRCP messages are communicated with the support of an XMPP framework. For this it is advised (bu t not mandatory) that every testbed deploys its own **XMPP server**.
- When receiving an FRCP message, the resource controller cannot just perform any action that is being requested. It has to verify if the entity that has sent him the XMPP message is actually authorized to request this action. For this the resource controller will contact the **Policy Decision Point** (PDP). This PDP will be able to make this authentication and authorization decision based on the information that was handed to it from the **Aggregate Manager** (which resource belongs to which slice, etc.). Note that this PDP is a new component in the architecture of cycle 2. It was introduced in order to protect testbeds from unauthorized control of its resources. Without it anyone that knows the XMPP topics to which the resource controller is listening, and has knowledge of the FRCP protocol that is being used, would be able to control the resource controller of that resource. In cycle 2 we upgrade from such a security-by-obscurity mechanism to a fully secure authentication and authorization scheme.
- The **experiment control server** is the entity that takes an experiment control scenario as an input, and processes it to define the specific moments when a certain action should be taken to the resources. It will be in charging for sending the corresponding FRCP messages to the appropriate resources at the correct time.

At the **experimenter** side, some experimenter tools can be applied for experiment control

- An **SSH client** allows the experimenter to login to the resources of the testbed himself, and to control the experiment manually.
- An **experiment controller** is identical to the experiment control server described above, but this time it is a local experimenter tool instead of a hosted tool. So the experiment controller takes an experiment control scenario as an input, and processes it to define the specific moments when a certain action should be taken to the resources. It is in charging for sending the corresponding FRCP messages to the appropriate resources at the correct time.
- The **scenario editor** is a tool that enables the construction of the experiment control scenarios that can be fed into the experiment controller or the experiment control server.

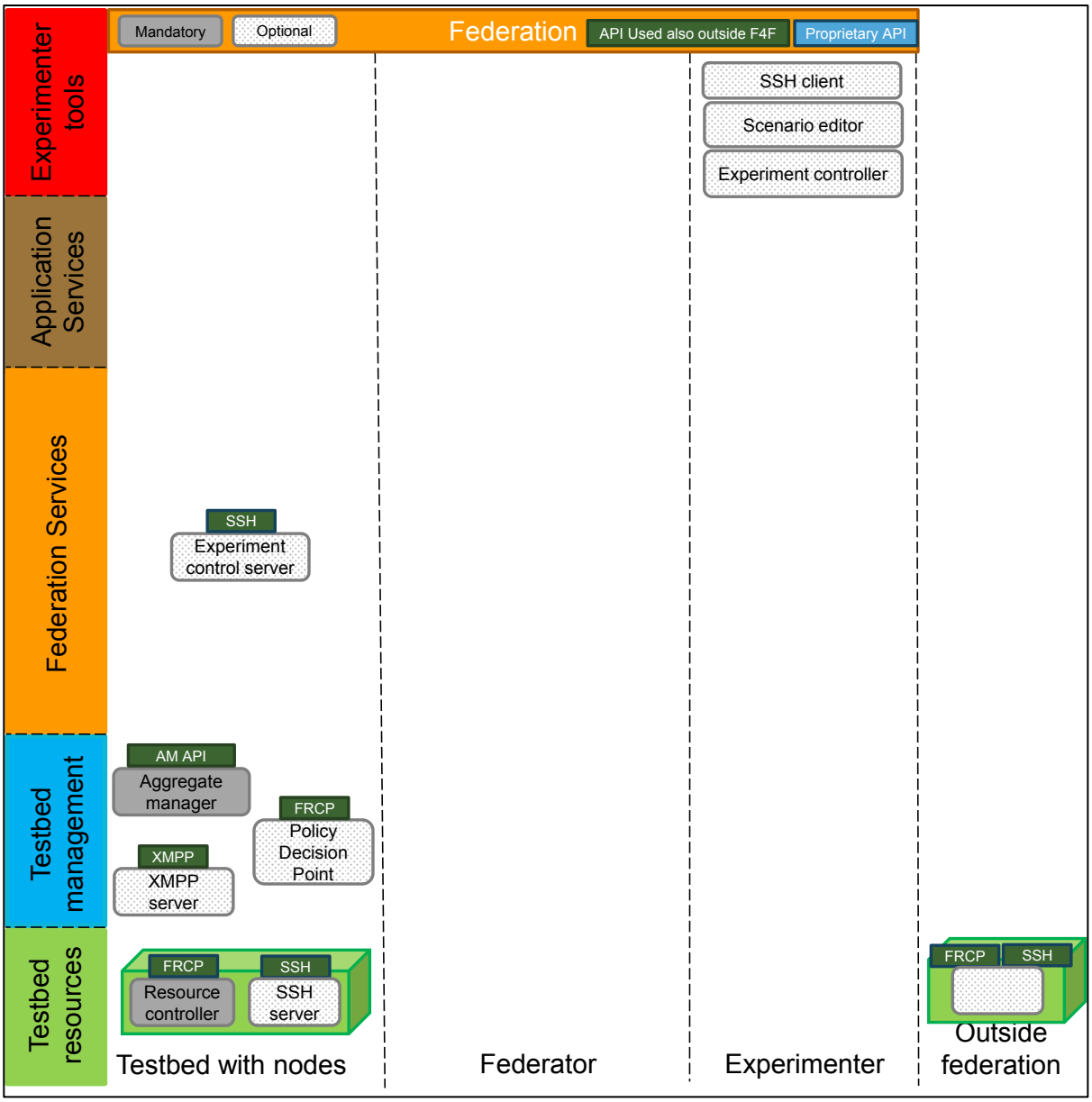


Figure 17: Cycle 2 architecture for Experiment Control

3.5.2 Sequence diagrams regarding experiment control

The simplest way to control an experiment is to log into the nodes with SSH, and to perform all corresponding actions manually. The corresponding sequence diagram is depicted in Figure 18. Note that it only introduces the corresponding interactions on a high level, more details should be given by WP5 and WP7 in their upcoming specification deliverables D5.2 and D7.2. The first step of login into the nodes with SSH is that during node provisioning, that the public SSH key of the experimenter is automatically copied from the AM to the resource (1). In the next step, the experimenter will login to the resource by connecting to its SSH daemon using his own SSH client (2, 3). Once connected, the experimenter can perform any action as he sees fit (4). Once finished, the experimenters logs out of this SSH session by disconnecting from the resource's SSH daemon (5, 6).

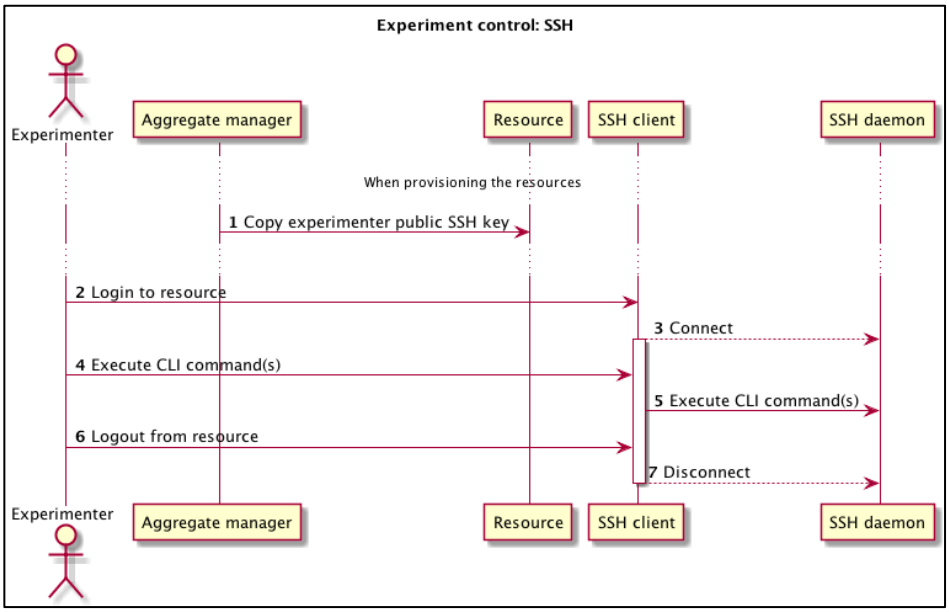


Figure 18: Sequence diagram for experiment control using SSH

The different steps involved in experiment control using an experiment controller are depicted in Figure 19. Note that it only introduces the corresponding interactions on a high level, more details should be given by WP5 and WP7 in their upcoming specification deliverables D5.2 and D7.2. When controlling the experiment with an experiment controller, first of all the Aggregate Manager programs the PDP with the needed information to allow it to make correct authorization decisions. More specifically, the AM informs the PDP about which resource belongs to which slice (1). The next step is then for the experimenter to define its scenario for experiment control (2), and to run it using its experiment controller (3). At due time, this will send the appropriate messages to the resource controller (RC) using the XMPP framework that is in place (4, 5). After receiving such a message, the RC will ask the PDP if it is OK for him to perform the requested action (6, 7). If so, the RC performs it (8). This is then repeated until the experiment control scenario has been entirely performed (9).

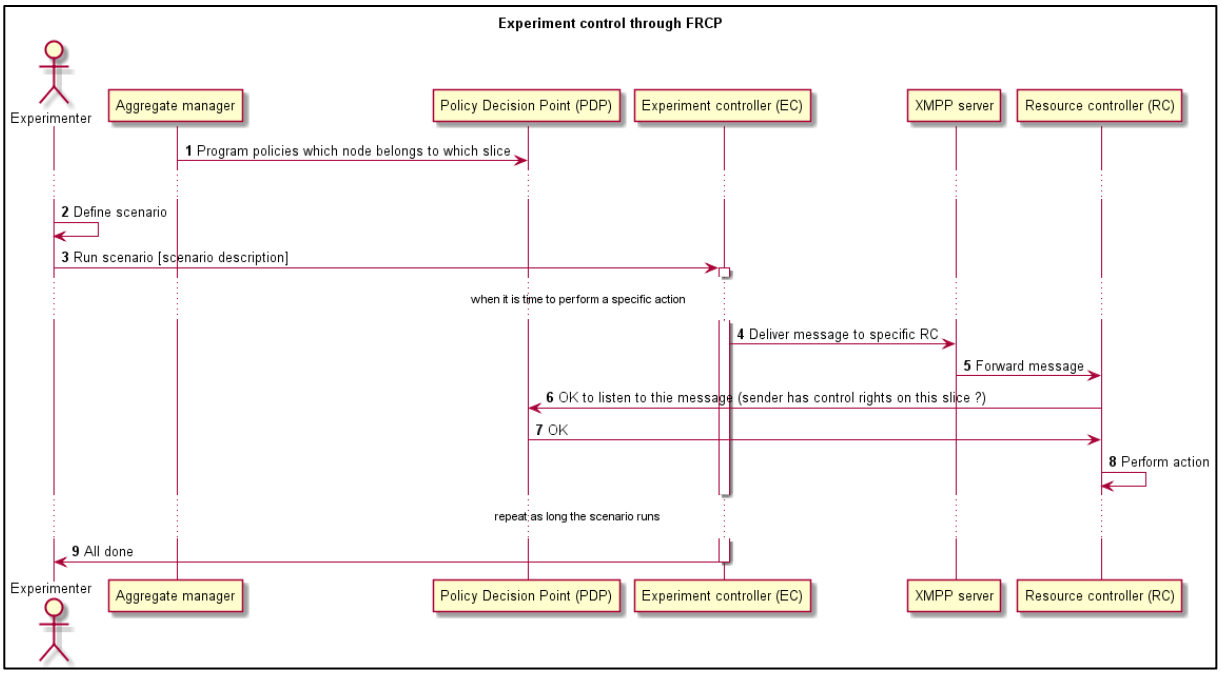


Figure 19: Sequence diagram for experiment control using an experiment controller

3.6 SLA management and reputation services

The part of the architecture that is presented here aggregates all architectural components needed to support SLAs and reputation services. The architecture itself is depicted in Figure 22. The remainder of this section will first introduce the SLA management lifecycle, discuss the adopted SLA management architecture and finally provide an explanation of the reputation service components.

One aspect that should be mentioned before diving deeper in the details of this part of the Fed4FIRE architecture is the fact that during the design it was intended to (if possible) base the Fed4FIRE SLA management solution on existing state-of-the-art SLA solutions such as e.g. Cloud4SOA [13]. Of course such a tool needs to be adapted and integrated with other Fed4FIRE architecture components, but it was perceived that this would definitely be more efficient then developing an SLA management framework from scratch.

To be a bit more specific regarding the Cloud4SOA project, this framework supports cloud-based application developers with multiplatform matchmaking, management, unified application, cloud monitoring and migration. It interconnects heterogeneous cloud offerings across different providers that share the same technology through the concept of adapter that provides a REST-based API for any cloud access. It is a clear contender to be selected as the starting point for the Fed4FIRE SLA management framework.

3.6.1 SLA Management lifecycle

As depicted in Figure 20, a typical SLA lifecycle can be split in the following phases (which can be grouped and simplified depending on the implementation):

1. *SLA Template Specification*: For a service provider, a clear step-by-step procedure describing how to write an SLA template to provide a correct service description.

2. *Publication and Discovery*: Publish the provider offer, the customer QoS needs, and possibility for the customer (experimenter) to browse/ compare offers.
3. *Negotiation*: Agreement on SLA conditions between the experimenter and the infrastructure providers.
4. *Resource Selection*: Depending on the chosen SLA, testbed providers select the resources that need to be assigned to the experimenter in order to meet this SLA. The resources can be explicitly selected by the experiment as well, depending on the testbed's policy.
5. *Monitoring and Evaluation of the SLA*: Comparing all the terms of the signed SLA with the metrics provided by the monitoring system, in order to internally prevent upcoming violations and to externally discover potential violations.
6. *Accounting*: This can cover different actions, such as invoking a charging/billing system according to the result, reporting, invoking a reputation system or an internal policy engine tool.

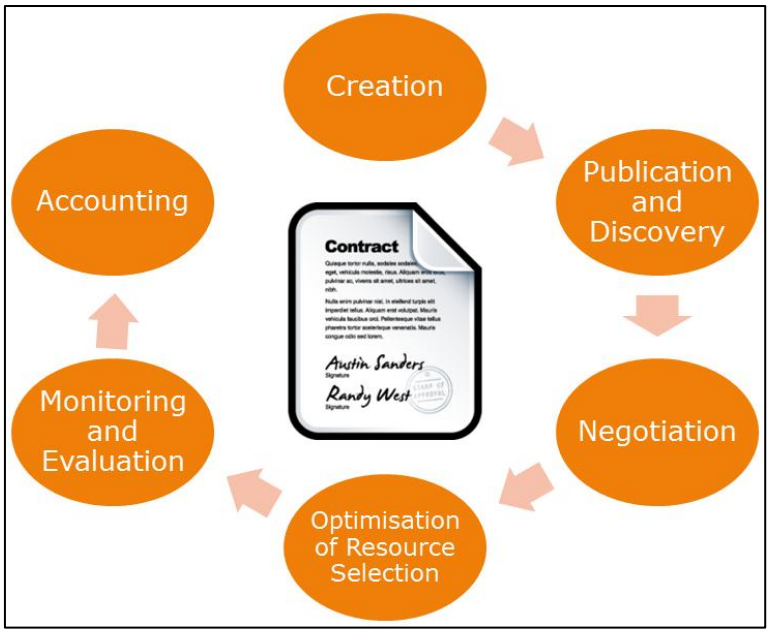


Figure 20: The SLA management lifecycle

3.6.2 SLA Management in Fed4FIRE

The first step of the architectural design of Fed4FIRE's SLA management framework was to evaluate the different possible architectural approaches. This analysis can be found in Appendix G. Based on the corresponding outcomes, the architecture could be defined. It is described in the remainder of this section.

3.6.2.1 Basic principles

In order to integrate SLA management in such a complex federation, Fed4FIRE has decided for cycle 2 to implement a simple SLA mechanism that is useful and understandable both by the community of experimenters and by the testbeds. This approach has been inspired by common SLA approaches in the state of the art (e.g. the Amazon EC2 SLAs [14], which guarantees the service is available during a fraction of the service time). This approach allows enabling simple SLAs that can be extended or enhanced by each testbed or by the federation later.

In cycle 2 we will start with two simple SLA mechanisms. This means we will implement them technically (we measure and report SLA violations) while the business case (what happens in case of violations) will not be considered apart from reporting overviews.

The **simplest mechanism** that will be implemented for all testbeds willing to adopt SLAs: based on the continuous measurements of the FLS dashboard together with the automated experiment and login tests, we can calculate a global uptime of a testbed.

A **more interesting mechanism for experimenters** is to have SLA feedback on their specific experiment. For this, we will calculate uptimes of resources in individual slices as follows. The SLA Type agreed to implement in iMinds' testbeds guarantees X% Uptime for Y% of the resources during the sliver lifetime.

An experiment is performed over a slice, which is a bundle of resources over one or multiple testbeds. A sliver is the part of that slice which represents a single offering of resources at one testbed. There would normally be one per testbed within a slice, but it may be the case that multiple slivers in a single slice are on the same testbed.

There will be one SLA between the experimenter and each of the different testbeds. However, the SLA enforcement will be performed per sliver as the RSpec Manifest (see 3.3.5) which is delivered by the AM API determines exactly which resources during which time are available to the experimenter. At the end of the experiment, the global information of the SLAs evaluation of all the slivers of the same testbed for the same experiment will be available. The lifecycle of these slivers could be parallel or consecutive.

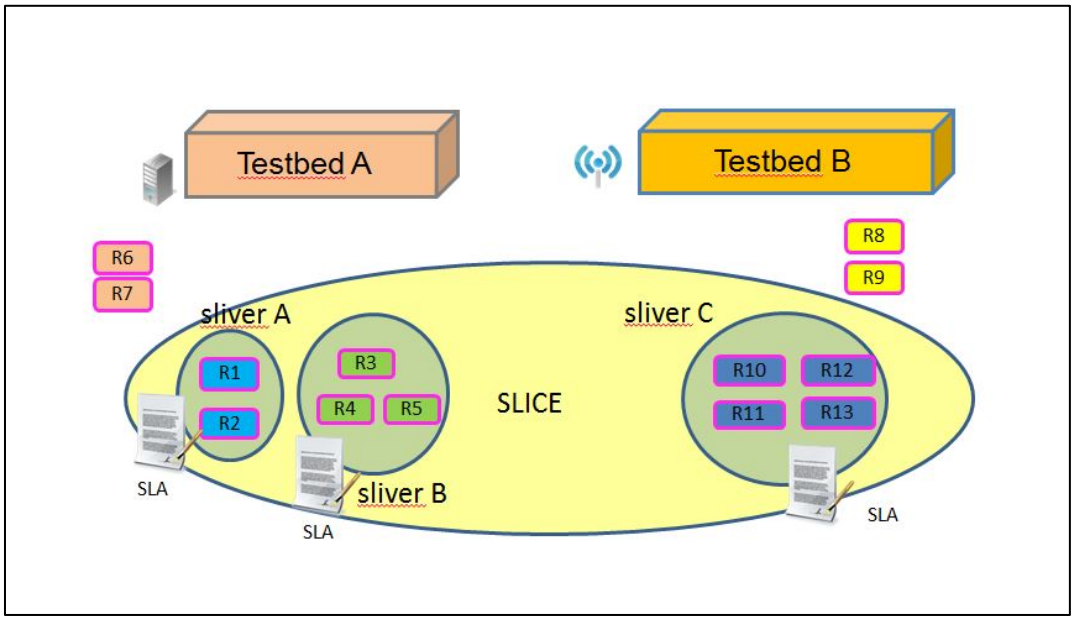


Figure 21: SLA agreements for different testbeds in one experiment

Some requirements needed in order to do the SLA Evaluation are:

- The Aggregate Manager (AM) should inform the SLA Management when the slivers have been provisioned or released and which resources are contained in them. Once the

resources of a sliver have been provisioned, they will not change during the entire sliver lifecycle. This is the Rspec Manifest.

- Once the Aggregator Manager has indicated the SLA Management module which resources are involved in the sliver, the SLA Management contacts the Monitoring System in order to provide the following data:
  - The identifiers of the resources involved in that sliver.
  - Metric to monitor the resources (availability).
  - And a time period of SLA monitoring.

- As soon as the SLA management begins receiving monitoring data, the enforcement starts (evaluation).

- The SLA Management is notified the sliver has been released by the AM. And it is only at this moment when the evaluation can be requested by the experimenter.

The evaluation of the SLA is calculated of the following way:

- The status of availability (UP, DOWN) of each resource will be stored by the SLA Management module at every SLA monitoring interval during the time it has been active.
- The uptime rate of each sliver is calculated of the following way:

  - The sum of all the availability values (UP = 1, DOWN = 0) obtained of each resource divided by the SLA monitoring interval during the time this resource has been active.
  - The SLA is met when the X% uptime rate defined in the SLA is fulfilled in at least Y% of the resources of the corresponding sliver.

- Finally, the SLA Evaluation of the different testbeds is shown to the experimenter, who might also request the SLA performance of each sliver for a specific testbed as soon as the slivers are released.

The testbeds of iMinds (w-iLab.t and Virtual Wall) will adopt the SLA mechanism implying commitments on individual resources within those testbeds (thus based on infrastructure monitoring). In case there is reservation of resources in place, the SLA will apply to the particular reserved resources. This can be seen as a pilot implementation of more elaborated SLAs that can be extended to other testbeds in the future.

Note that SLAs on reservations in the future are similar if you consider the SLA evaluation per slice. At the end of the slice-time, one can evaluate if the uptime/availability of all promised resources was met.

3.6.2.2 *SLA architectural components*

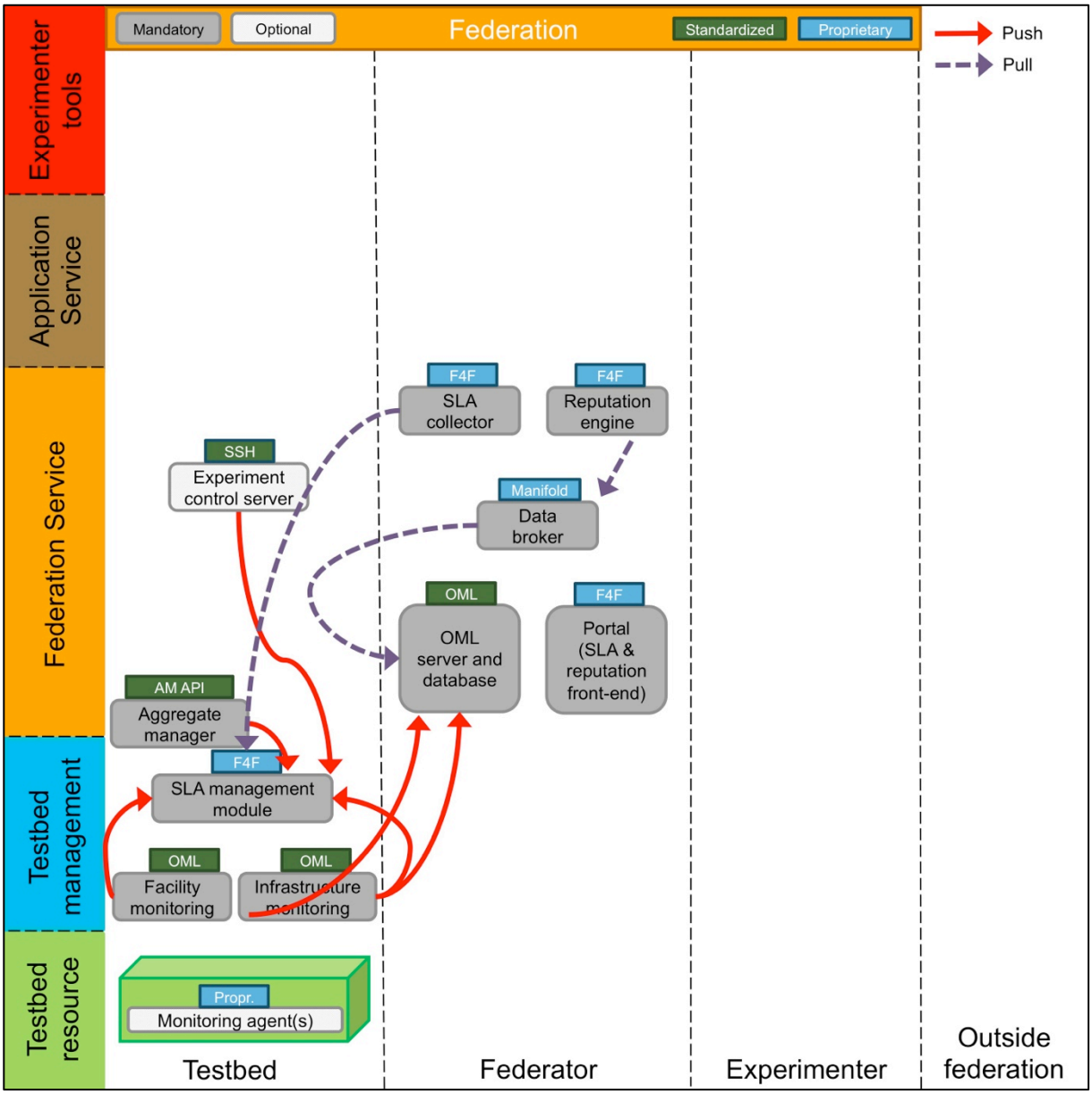


Figure 22: SLA and reputation architecture

As shown in Figure 22, the distributed SLA architecture is composed of three main parts:

- **SLA front-end tool**: the goal of this component is to show the functionalities of the SLA Management to the experimenters through a Graphical User Interface. The SLA front-end tool is also responsible to do the correct request to the SLA Collector and then gather and show the results. To implement the SLA Front-end tool, some plugins will be developed and integrated in the Portal and in the different front-end tools. As shown in Figure 21, the SLA Front-end tool is not envisaged to be a standalone tool, but it is considered to be provided as an integrated part of the portal.

- **SLA Management Module**: this component is responsible for supervising if all the agreements reached are respected. It receives and processes all measurements related to the SLA from the monitoring system. It validates whether the measurements are within the thresholds established in SLA agreement metrics. In case the testbed does not fulfill these conditions during the execution of the experiment, it triggers the appropriate actions in case of violations. The SLA is evaluated during the sliver and thus, the SLA Management module needs to be aware of the start and end times of the sliver. For this, it will be in communication with the Aggregator Manager (AM) of the testbed where it runs. For infrastructure monitoring-based SLAs (iMinds testbeds in cycle 2), there will be a relation between the AM and the SLA Management module. The AM will notify when the provisioning of the resources has taken place and which resources have been reserved.
- **SLA Collector**: this component acts as a broker between the different client tools and the different SLA Management modules of each testbed. The goals of this component are:
  - Gather the different warnings and the evaluations when the experimenter wanted.
  - The SLA collector can also provide SLA information to the reputation service if required.

3.6.2.3 Illustration of the Fed4FIRE SLA workflow

The SLA workflow is depicted in the following picture. The main concept is that the experimenter will accept the SLA after selecting the resources (browsing the testbeds offering SLAs is also possible) and, once the resources are provisioned, the SLA evaluation is based on monitoring information for those particular resources. Once the sliver is released, he experimenter can request the SLA evaluation and, for those slivers having failed, the specific violations can be shown.

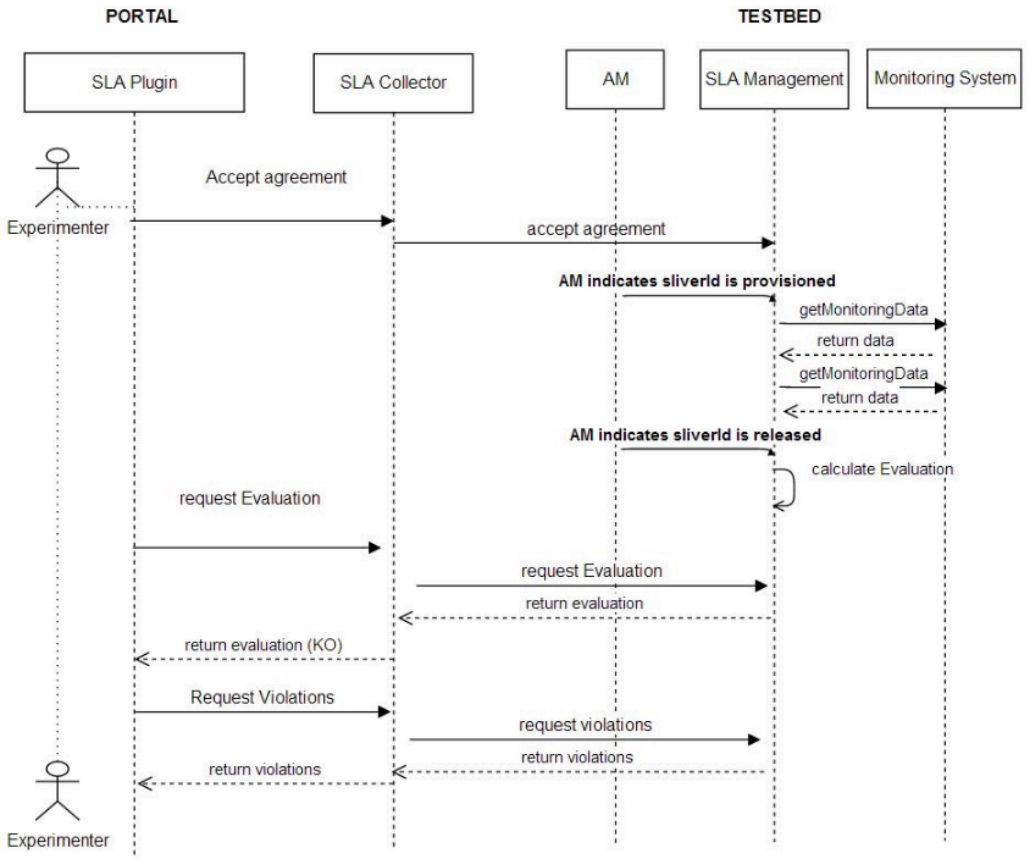


Figure 23: The SLA management lifecycle

3.6.3 Reputation Service in Fed4FIRE

The reputation service in Fed4FIRE aims to provide mechanisms and tools towards building trustworthy services based on the combination of Quality of Experience (QoE) and monitoring data. The developed mechanisms and tools will reflect the end users (experimenters) perspective with the objective of empowering the users/experimenters to select testbeds based on dynamic performance metrics. These metrics will offer a "smart" user support service that provides a unified and quantitative view of the trustworthiness of a facility.
In order to achieve that, the service will mainly focus on building reputation-based trust utilizing:

1. Raw monitoring data (e.g. information to experimenters on up-time, usage etc. that results into site popularity) and SLA information
2. Users' feedback regarding their Quality of Experience (QoE) and service received.

The main interactions of this service (depicted as the **reputation engine** on Figure 22) with the other Fed4FIRE components include communication with: (1) the future reservation broker to retrieve data regarding the resources used in user's experiment, (2) the monitoring data broker from which measurement data for the reserved resources will be obtained with respect to user's experiment (3) the SLA service from which will be obtained SLA information regarding violations, etc. and (4) the Portal which will serve both as a place for displaying testbeds' reputation scores (statistics) and as a feedback page for users' Quality of Experience (depicted on Figure 22 as the **reputation front-end**, which is an integrated part of the portal).
More information regarding the reputation service is provided in deliverable D7.2.

4 Main differences with cycle 1 architecture and D2.1

Throughout this deliverable, the description of the second federation architecture of the project has been intentionally described as something on its own, without referring to how the architecture looked like before. The reason for this is that we wanted to make it easy for people novel to the project to understand exactly how our current architecture looks like and operates. We feared that focusing on the delta's between cycle 2 and cycle 1 would possibly confuse the reader.

However, we are aware that for people that have been involved in the project since its very beginning, it is interesting to get a clear view on how the architecture for cycle 2 compares to that for cycle 1. The most important differences are therefore listed below:

- Definition of member and slice authority instead of identity provider
- Definition of federation model and introduction of the concepts outside of the federation
- Definition of slice and sliver
- Removal of the brokers layer and introduction of the federation services layer and application services layer
- Federator instead of central location(s)
- Definition of currently three authorities in the Fed4FIRE federation
- Rename certificate directory to authority directory
- Introduction of documentation service in the federator
- Rename testbed directory to aggregate manager directory
- The cycle 1 architecture did not capture some finer details defined by WP5 regarding experiment control: it does not indicate that FRCP is the adopted API for this functionality, and that an experiment controller is to be foreseen on every resource. Also, this adopted approach needed to be made more secure when transferring it from the single-testbed to the testbed federation domain.
- Similarly, the first architecture did not yet capture the decision to implement facility monitoring by exporting the corresponding data as OML streams, and collecting them in a central OML server that is queried by the FLS dashboard.
- As requested by the WP4 community, services were contemplated. This resulted in the addition of the application services layer next to the federation services.
- Even in cycle 1 some architectural components were optional, others were mandatory. However, the cycle 1 architecture didn't depict them in a different way.
- In experiment control, the PDP component was introduced.
- Sequence diagrams were introduced to show the detailed workflow of the particular parts of the architecture (on some parts D2.1 was lacking these details).
- SLA management and reputation services were introduced.
- A detailed setup was introduced for the First Level Support dashboard.
- Detailed technical examples of the concepts of delegation, handover between AM and PDP and subauthorities as a base for distinction between the experimenters have been introduced in cycle 2 (appendixes D, E and F).
- Description in detail of the workflows for setting up an account, creating SSH keys and doing a first experiment are included (appendices G, H and I).

5 Requirements which are fulfilled with the architecture in cycle 2

This goal of this chapter is to analyze to which degree the federation architecture for the second cycle of the projects meets all requirements and constraints imposed on it in chapter 2. This allows us to assess where we are with the project, and to identify the issues that must definitely be addressed in cycle 3. In order to keep the length of this deliverable within reasonable limits, it was chosen to include this analysis formation into the different related appendices:

- Appendix A: Requirements from the infrastructures community (D3.2)
- Appendix B: Requirements from the services community (D4.2)
- Appendix C Requirements from SLA management

To annotate the degree in which a requirement is fulfilled in this cycle 2 architecture, the requirements in the appendix have been color coded to show which are fulfilled by the architecture and which are not:

- In green, the requirements which will be fully fulfilled by the proposed architecture
- in orange the requirements which are partially fulfilled
- in white the requirements which are not yet fulfilled.

In the table below, the counts of these color-coded annotations are summarized, in order to give a quick overview of the big picture in terms of covered requirements in cycle 2. Note that for some sources the requirements were presented in a more aggregated fashion, resulting in lower absolute numbers in terms of total requirements (e.g. WP4 did this for its requirements on experiment workflow and lifecycle management, measurement and monitoring, trustworthiness and interconnectivity). Other inputs have chosen to present their requirements in a much finer-grained fashion (e.g. the SLA requirements), resulting in much higher totals. So comparing the absolute numbers between requirements categories does not make much sense. But the spread per category gives a good estimate of how well certain aspects of the overall desired Fed4FIRE federation framework are tackled today.

Table 1: Overview of how well the different requirements are fulfilled by the architecture of cycle 2

Requirements category	Total # requirements	# covered	# partially covered	# not covered
Experiment workflow and lifecycle management	28	22	4	2
Measurement and monitoring	9	7	0	2
Trustworthiness	6	6	0	0
Interconnectivity	6	4	1	1
SLA requirements	29	17	0	12

6 Conclusion

This second deliverable of task 2.1 defines the architecture for cycle 2 of the Fed4FIRE project.
It takes as input requirements of WP3 (Infrastructure community), WP4 (service community), WP8 (First Level Support), SLA management and task 2.3 (sustainability) and defines an architecture that copes with as much requirements as possible and will be implemented for cycle 2 of Fed4FIRE. It is an evolution based on the cycle 1 architecture.

In chapter 2 all these requirements are listed, while in chapter 3, we describe the architecture for the different steps in the experiment lifecycle (discovery, specification, reservation, provisioning, monitoring and measurement, experiment control and SLA management). For cycle 2, just as in cycle 1, discovery, specification and provisioning will be done based on the SFA GENI AM API v3 standard, while for advanced reservation and extended policy based authorization extensions are currently being made. Regarding the RSpecs, in cycle 2 each testbed can still provide its own RSpec to be used and tools have to be adapted to these RSpecs.

The architecture defines also multiple identity providers with a chain of trust, and a central portal accompanied with an identity provider and directories (machine and human readable) for tools, testbeds and certificates.

For First Level Support, the architecture defines facility monitoring which should be identical for all testbeds and should make it possible for first level support to have a high level overview of testbed and resource availability.

For monitoring, three types of monitoring have been identified and described. Facility monitoring is used to monitor a testbed as a whole and is e.g. interesting for First Level Support and experimenters to know if a testbed is functioning correctly or not. Infrastructure monitoring is monitoring information which is provided by the testbed itself to an experimenter (e.g. switch port statistics, spectrum measurements or virtualization infrastructure monitoring) which are normally not available to an experimenter. Experimenter monitoring is then monitoring the experimenter can do on the nodes themselves.

For SLA management, a first architecture is introduced which is based on monitoring information coming from the testbeds. This should make it possible to gain experience with simple SLA evaluation.

Chapter 3.6.3 highlights the main differences with the cycle 1 architecture and D2.1.

Chapter 5 lists a short summary about the requirements which are fulfilled and which are not, and referes to the appendix for the detailed list.

As a general conclusion, this 2nd cycle architecture is an evolvement of cycle 1, with lots of improvements, clarifications and additions on multiple points, but the basic ideas remain the same, which means that implementation and deployment do not have to change radically but will be further go to unified interfaces and workflows.

References

[1] B. Vermeulen, W. Vandenberghe, T. Leonard, et al. “D2.1 – First federation architecture”, deliverable of the FP7 Fed4FIRE project, December 2012

[2] C. Scognamiglio, M. Sioutis, S. Bouckaert. “D3.2 – Infrastructures community federation requirements, version 2”, deliverable of the FP7 Fed4FIRE project, December 2013

[3] F. Lobillo, M. Saywer, G. Francis, et al. “D4.2 – Second input from the Services and Applications community to the architecture”, deliverable of the FP7 Fed4FIRE project, November 2013

[4] D. Davies. “D8.4 – Second input to WP2 concerning first level support”, deliverable of the FP7 Fed4FIRE project, September 2013

[5] J. Van Ooteghem, B. Naudts, W. Vandenberghe et al. “D2.3 – First Sustainability Plan”, deliverable of the FP7 Fed4FIRE project, July 2013

[6] “GENI Aggregate Manager API Version 2 details”, http://groups.geni.net/geni/wiki/GAPI_AM_API_V2_DETAILS, last accessed February 3rd 2014

[7] “GENI Aggregate Manager API Version 3”, http://groups.geni.net/geni/wiki/GAPI_AM_API_V3, last accessed February 3rd 2014

[8] “Operations on Individual Slivers”, http://groups.geni.net/geni/wiki/GAPI_AM_API_V3/CommonConcepts#OperationsonIndividualSlivers, last accessed February 3rd 2014

[9] L. Peterson, R. Ricci, A. Falk and J. Chase. “Slice-Based Federation Architecture”, available for download on http://groups.geni.net/geni/attachment/wiki/SliceFedArch/SFA2.0.pdf, last accessed February 3rd 2014

[10] “GENI Design Activities”, http://groups.geni.net/geni/wiki/GeniDesign, last accessed February 3rd 2014

[11] “GENI RSpec version 3”, http://www.geni.net/resources/rspec/3/, last accessed February 3rd 2014

[12] “Clearinghouse”, http://groups.geni.net/geni/wiki/GeniClearinghouse, last accessed February 3rd 2014

[13] E. Kamateri, N. Loutas, D. Zeginis, et al. “Cloud4SOA: A semantic interoperability PaaS solution for multi-Cloud platform management and portability”, Service-Oriented and Cloud Computing Lecture Notes in Computer Science, Volume 8135, pp 64-78, Springer, 2013

[14] “Amazon EC2 Service Level Agreement”, http://aws.amazon.com/s3-sla/ , last accessed February 3rd 2014

Appendix A: Requirements from the infrastructures community (D3.2)

Experiment Workflow and Lifecycle Management

Req. id	Req. statement	Req. description	Comments
I.1.101	Node capabilities	Fed4FIRE must provide a clear view on what node capabilities are available, and this should be defined in the same way across the federation. This view should be returning all the nodes that are offered by the testbeds, and should not filter out those that have been reserved for now. This should also not only include hardware characteristics such as CPU type or available types of network interfaces, but should also contain information about other capabilities such as is it mobile, can it be accurately steered remotely, etc. If resources have a static relation with each other (e.g. node X is installed onto mobile robot Y) then this should also be represented.	Node capabilities can be described in terms of CPU architecture and speed, RAM, supported 802.11 standards, optical networking interfaces, software defined radio, measurement resource type, OpenFlow support, etc. It can be beneficial to adopt proven standards to represent these capabilities (e.g., FOAM which provides a comprehensive OpenFlow resource description).
I.1.105	Discovery through federation-wide APIs	Resource discovery must be integrated into uniform tools through federation-wide APIs. Ideally, these APIs would be compatible with discovery APIs already supported by the infrastructures and/or existing uniform tools. This would decrease the development costs for the infrastructure providers and tool builders.	The APIs supported by all infrastructures in the Fed4FIRE federation should be able to support both the Fed4FIRE portal and any other standalone tool that wishes to adopt them. Therefore the APIs should be well documented.
I.1.106	Intra-infrastructure topology information	For nodes that have wired and/or wireless network connections to other nodes within the same testbed, it should be possible to identify the physical topology. This relates to connections which are part of the data plane of an experiment, not the control interfaces. Similar, if virtualized topologies are supported, the corresponding possibilities should also be communicated to the experimenter. As a result, it should be easy for experimenters to assess the scale of the testbed.	Examples of wireless topologies are Wi-Fi or 802.15.4 connectivity charts (possibly with variable channel and modulation type selection), or connectivity map between WAN base stations and nodes. Examples of virtualized topologies are those based on wavelength allocations in the optical domain, or those based on VLAN configurations in Emulab.
I.1.107	Inter-infrastructure topology information	It should be known how different infrastructures are/can be interconnected. Important parameters are the type of interconnection (layer 2, layer 3), and the support for bandwidth reservation. If	

Req. id	Req. statement	Req. description	Comments
		resources are also reachable beyond the boundaries of the Fed4FIRE partners' infrastructures (e.g., because they are directly connected to the public Internet), this should also be mentioned. Information regarding IPv6 support on the inter-infrastructure topologies is also required.	

Req. id	Req. statement	Req. description	Comments
I.1.201	Manually extract requirements from discovery query results	When the query in the discovery phase returns a certain list of resources, it should be possible for the experimenter to select the resources he/she would like to include in the experiment. This should be supported in relation with a specific resource ID (e.g., I want this specific node at this specific Wi-Fi testbed).	

Req. id	Req. statement	Req. description	Comments
I.1.301	Hard resource reservation	Fed4FIRE must provide hard reservations of the available resources. It should be able to perform immediate reservations (starting from now), or more advanced reservations (given a specific future timeslot that the experimenter would want, or have the reservation system look for the first available slot where all desired resources are available).	It should even be possible to reserve all nodes that could interfere with an experiment, even if they are not actually used during the experiment. E.g., the experimenter could not trust channel reservation, since in reality orthogonal channels often do interfere due to imperfect radio hardware implementations. Therefore he/she wants to reserve all nodes in a specific infrastructure for the experiment. Another example resource for which hard reservation would be indispensable is that of wavelengths in optical OFELIA resources.
I.1.302	Fairness	A means to enforce fairness with hard reservations is required. Situations where a few users reserve all nodes for too long should be avoided. Similarly, situations where a large amount of users reserve a few resources for a very long time should also be avoided. This kind of reservations is typically done to develop new solutions on the testbed, but makes it harder to schedule other large-scale experiments.	This could be achieved by specifying an expiration date for reservations via calendar or a scheduler, or through the usage of reservation quota. It could be interesting to look at existing techniques applied in high performance computing clusters.

Req. id	Req. statement	Req. description	Comments
I.1.303	Secure reservation	Fed4FIRE must provide a reservation system with adequate security to provide assurance to industrial users	
I.1.304	Automated reservations handling	The Fed4FIRE reservation system should be able to approve/deny reservation requests in a fully automated manner, without any manual intervention by the infrastructure operators.	

Req. id	Req. statement	Req. description	Comments
I.1.401	Provisioning API	APIs are required to enable direct instantiation of both physical and virtualized resources for experiments. Ideally, these APIs would be compatible with provisioning APIs already supported by the infrastructures and/or existing uniform tools. This would decrease the development costs for the infrastructure providers and tool builders.	Instantiation of physical node would involve powering the node on, and appropriately steering the node boot process. Instantiation of virtualized resources can be related to the setup of virtual machines, OpenFlow flows, etc. For such virtualized resources the API should support an annotation mechanism to define the actual physical resource on which the virtual one should be instantiated.
I.1.402	Customizing Linux	Fed4Fire must provide the ability to install a specific custom Linux kernel or distribution on the nodes	Experimenters could e.g., require specific Linux kernels because some experimental hardware drivers might only be supported on such a specific kernel. It can also allow them to deploy specific Linux distributions, e.g., OpenWrt. Infrastructure providers could assist their experimenters by providing several pre-installed Linux distributions (Some Ubuntu Long Time Support versions, Fedora distributions, OpenWrt, etc.) to choose from.
I.1.403	Root access	Fed4FIRE must provide the possibility to access a node as root user	Often experimenters will install additional software on their resources. This can be external software packages, compiled code, new hardware drivers, and so on. To do so, root access to the node is required. In many cases the experimenters also want to configure the network interfaces according to their experimentation needs. This also requires root access.
I.1.404	Internet access to software package	In Fed4FIRE software installation through a packet manager (e.g., apt-get) must be possible. Hence the package manager should have Internet access to external	

Req. id	Req. statement	Req. description	Comments
	repositories	software package repositories.	
I.1.405	Hard disk imaging	Once experimenters have finished the configuration of their nodes, they should be able to create a binary image of the entire hard disk drive, which can be stored and reloaded to the node in a future experiment. This allows to setup experiments which need a longer runtime, but pause its execution from time to time to allow other experimenters to also use the testbed in case of hard reservation of the resources.	Infrastructures can operate perfectly without HDD imaging support. In that case experimenters have to write appropriate installation scripts that are automatically started at boot-time.. However, manually installing all software once on a node, and creating an image from that prototype is more convenient/efficient. HDD imaging support also allows infrastructure operators to provide pre-configured images that include specific valuable functionality. An example would be a fully configured GNU radio image that can be flashed to nodes connected to a SDR.
I.1.407	Automated software installation at boot time	In advance you should have been able to already define what these specific resources should do at startup (install additional software, copy specific files, start specific daemons/tools)	

Req. id	Req. statement	Req. description	Comments
I.1.501	SSH access	Nodes must be accessible via SSH.	This is most valuable during the development phase or for debugging purposes.
I.1.502	Scripted control engine	It must be possible to describe advanced experiment scenarios by the use of a script that will be executed by a control engine. The engine will perform all required shell commands on the appropriate resources at the appropriate time. Ideally, this control engine would be compatible with engines already supported by some of the infrastructures. This would decrease the development costs for the infrastructure providers.	This way the experimenter can alter the behaviour of the resources in an automated manner from a single location, without having to manually login on all nodes during experiment runtime. This is not only more convenient, but also increases repeatability and hence scientific value of the experimental results. It also allows the quicker setup of complex experiments at different infrastructures.
I.1.504	Generality of control engine	The experiment control engine should be general enough to support the control of all possible kinds of Future Internet technology: wireless networks, optical networks, OpenFlow devices, cloud computing platforms, mobile robots, etc. If it is not feasible to control all these	

Req. id	Req. statement	Req. description	Comments
		resource types with a single tool, you should at least be able to use the same tool per domain (e.g. one tool for all cloud computing tasks, one for all wireless tasks, and one to control the robots movements).	

Measurement & Monitoring

Req. id	Req. statement	Req. description	Comments
I.2.101	Measurement support framework	Fed4FIRE must provide an easy way for experimenters to store measures during the experiment runtime for later analysis. The data should be clearly correlated to the experiment ID.	Measurements can be related to common metrics for which existing tools such as ping or iperf can be used. However, they can also be very specific to the experiment, and hence calculated somewhere within the experimental software under test. It should be possible to take measurements on a large variety of resources: Linux servers/embedded devices, OpenFlow packet switches and optical devices, cellular base stations, etc.
I.2.102	Automatic measurement of common metrics	Common characteristics should be stored automatically during an experiment (CPU load, free RAM, Tx/Rx errors, etc.)	.
I.2.103	Wireless interference	Information about external wireless interference during the execution of the experiment should be provided.	The interference can be detected using monitor interfaces in dedicated nodes and/or spectrum analysers that offer more exact results. This functionality should be easily provided to every experimenter who is not “spectrum analysing” expert.
I.2.105	Monitoring resources for suitable resource selection and measurement interpretation	Fed4FIRE must also provide the monitoring info regarding the state of the resources to the experimenters. This way they can choose the best resources for their experiments. This information also provides the experimenters with the means to distinguish error introduced by the experiment from errors related to the infrastructure.	In this monitor view that an experimenter has on his/her resources, it could also be interesting to display some non-monitored background information, for instance the IP address of the control interface, the DNS name, etc.
I.2.106	Minimal impact of monitoring and measuring tools	As less overhead as possible should be expected from the monitoring and measurement support frameworks. The impact of the measurement tools over the experiment results should be negligible.	

Req. id	Req. statement	Req. description	Comments
I.2.107	On-demand measurements	The user must be able to request on-demand measurements. In order to do so, they will need to express that they want agents with such on-demand polling capacities	The same information can be retrieved by looking into the output of the monitoring and measurement tools that will continuously provide measurements during the experiment run-time. However the on-demand measurement is more convenient during experiment development and debugging.

Req. id	Req. statement	Req. description	Comments
I.2.202	Data security	Access to the data should be properly secured	
I.2.203	Stored experiment configuration	Experiment configurations should be stored in order to replay experiments and compare results of different runs. These configurations should be versioned in a way that corresponds with significant milestones in the experiment development.	Experiment configurations can contain deployment descriptors, experiment control scripts, etc.

Trustworthiness

Req. id	Req. statement	Req. description	Comments
I.3.101	Single account	Fed4FIRE must provide the mean of accessing all testbeds within the federation using one single account (username/password). Ideally, this authentication framework would be compatible with those already supported by some of the infrastructures. This would decrease the development costs for the infrastructure providers.	This means both accessing the web interfaces of the federated infrastructures, as accessing the actual resources belonging to the experiment, retrieving the experiment results, and so on.
I.3.102	Public keys	Fed4FIRE should also provide authentication by the use of public SSH keys.	It is possible that for some resources it is technically more feasible to authenticate through public SSH keys. Therefore Fed4FIRE should not only provide the single account based on username/password, but also on a pair of public/private keys.
I.3.104	Authentication of API calls	Access to the Fed4FIRE APIs (discovery, reservation, provisioning, etc.) should also be protected by an authentication mechanism	

Req. id	Req. statement	Req. description	Comments
I.3.104	Low barrier to create a Fed4FIRE identity	It should be easy for new experimenters without any affiliation to the federation to create their Fed4FIRE identity.	

Interconnectivity

Req. Id	Req. statement	Req. description	Comments
I.4.001	Layer 3 connectivity between testbeds	The resources within a Fed4FIRE infrastructure should be able to reach the resources deployed in the other Fed4FIRE infrastructures through a layer 3 Internet connection.	Ideally all infrastructures are connected to high-capacity research Internet backbones such as Géant.
I.4.003	Transparency	Providers must be able to offer in a transparent way the resources of all the federated testbeds. Interconnectivity solutions should not introduce unneeded complexity in the experiment.	Solutions based on VPN or other tunnels require that the experimenter is aware of the corresponding configurations when developing the experiment, while he/she should be concentrating on the content of the experiment, and not these practical preconditions. Besides, VPN tunnels will work initially, but they will not scale when a larger number of infrastructures has to be interconnected, due to conflicts in address spaces.
I.4.004	Per-slice bandwidth reservation	Per experiment, Fed4FIRE should provide the possibility to reserve bandwidth on the links that interconnect specific infrastructures.	
I.4.005	IPv6 support	The ability to conduct IPv6 measurements and to interact with the nodes of other testbeds over IPv6 should be enabled.	After interaction with its research community, iMinds reported that IPv6 support is crucial since some of the testbeds (e.g. Virtual Wall and w-iLab.t) need to support IPv6 due to a shortage of public IPv4 addresses and the need for transparent interconnectivity. If resources of the other testbeds need to interact with these IPv6-based testbeds, it is a must that these other testbeds also support IPv6, or no communication between the resources will be possible. Because of this element, it was

Req. Id	Req. statement	Req. description	Comments
			chosen to upgrade the priority of this requirement to high
I.4.006	Information about testbed Interconne-ctions	The experimenter needs to know how the several testbeds are interconnected e.g., via layer 3 or layer 2. Especially he/she needs to know which gateways should be used by the resources in order to interconnect them along with other testbed resources. The experimenter also wants to know the type of interconnection between the testbeds used in its experiment (dedicated direct links, interconnected through Géant using best effort, interconnected through Géant with bandwidth reservation, connected to the public internet but not to Géant, ...)	

Appendix B: Requirements from the services community (D4.2)

Experiment Workflow and Lifecycle Management

Req. id	Req. statement	Req. desc.	Comments	Justification of Requirement
ST.1.001	Unified interface for register external resources	Fed4FIRE should provide a standardised way of registering and unregistering external devices. This way, experimenters will only have to communicate with this component/interface to cope with the registration and deregistration of the external resources. External resources can be dynamic (e.g. portable devices that might be turned on and off) and include different devices, from a smartphone, tablet, to a dedicated user hardware etc. The experimenter also requires up to date information of the status of this kind of devices, since end-users might turn them on and off at their will. The fact that this kind of resource appears and disappears might not alter the course of the experimentation. When these devices appear, they might become part of the infrastructure (e.g. as sensing nodes and thus producing information). This also applies for VMs, for example, since experimenters should be able to dynamically create VM (resources), name them and use that name to interact with them.	It relates to registration of resources in testbeds, specification of resources, advertising	This requirement is important in case external resources such as mobile devices are involved in an experiment as end-users. A standardised interface for registering resources will enable all testbeds to register more easily than if they had to learn many different registration procedures. The experimenter must be aware of whether a concrete mobile device is participating or not at a given time
ST.1.002	Dynamic scaling of experiment resources	During an experiment, the experimenter might need to decide if and how to change the amount of computing or network resources allocated to a service. For example, scalable network provisioning with the ability to control the bandwidth available to the experiment. Once running, and even at short notice, we need to have a way to scale up and down (e.g. more VMs, fewer VMs). Switching from one computing supplier to another is	This could imply scaling resources - adding and removing them from testbeds or adding / removing a whole testbed	This is useful in cloud-related and/or network capacity experiments. It is possible that some experiments will need to change the resources they use as the experiment is running.

Req. id	Req. statement	Req. desc.	Comments	Justification of Requirement
		also required.		
ST.1.003	Experiment across multiple testbeds using homogeneous description of resources	Fed4FIRE must provide tools to deploy an experiment over a subset of federated testbeds. Resources must be described in a homogeneous manner so that the experimenter can compare the experiments in different testbeds, giving the experimenter a view into the internal steps the provisioning of his resources go through.	Fed4FIRE must able to deploy an experiment on multiple testbeds selected by the experimenter.	This requirement relates to orchestration capabilities, easing the process of experiment provisioning across several testbeds. This is useful so the experimenter can compare different testbeds' resources. This visibility helps the user understand issues he might be having. This gives important feedback when operation takes time. Resources must be advertised and specified so the experimenter knows what to expect
ST.1.004	A variety of resource reservation options for scheduling parallel processing with limited dynamic scaling	Experimenters must be able to reserve and later access resources on more than one testbed at a time and run parallel processes. The experimenter will have multiple resource reservation options to enable testing of different resource planning strategies against different criteria (e.g. costs, reputation, knowledge, etc.) If reservation is in place, rather than the assumption of infinite (or at least sufficient) elasticity Fed4FIRE must enable to possibility to set limits.	Scheduling simultaneous access to multiple testbeds. This requirement is also related to resource management on testbeds, SLA management, enforcing limits, making experimenters accountable for the resources they request and use.	This is important in cases where the experiment demands it, for example parallel processing over two testbeds, or scheduling processing and network resources so the process results can be transmitted

Req. id	Req. statement	Req. desc.	Comments	Justification of Requirement
ST.1.008	Automatic processing triggers based on sensing context and environment	Some features could be executed automatically. For example, a reserved experiment could start automatically. Also, during the experiment, decisions can be autonomously taken by the federation depending on contextual information (location of the service user, thresholds, etc.)		If a reservation starts in the middle of the night, there must be a way for the experimenter to have his experiment ran automatically.
ST.1.010	Multiple testbeds available through a common set of tools	Multiple testbeds of different kinds federated together in one experiment through a set of common tools so that the overhead effort required to use the testbeds gets minimised and the experimenters can focus on the testing itself. For example, computing resources and network resources connected together.	Data sharing, connectivity and experiment management across multiple testbeds - Data management & migration of data	This is the essence of Fed4FIRE and the reason for building a federation
ST.1.013	User friendliness	In general, Fed4FIRE tools should be user friendly to the experimenter		The complexity of the infrastrcuture technical details should be minimised as much as possible for the experimenter
ST.1.014	Software exposure and use	Packing up resources in services offered to the experimenter so that he/she is isolated from the real infrastructure (e.g. ready-to-use network behaviour packages: high congestion, medium congestion, small congestion). Fed4FIRE must provide the means for these applications to be exposed, discovered by experimenters, provisioned and used. For example, experimenters should be able to generate load (from thin clients, using traffic generators, etc.), using existing tools available at testbeds or provide the possibility to gather information from social media (could be simulated). Very importantly, Fed4FIRE will be able to support (i.e. offer, provision, monitor, etc.) low level resources services (hardware) and, at the same time, high-level application services (software). It will be up		

Req. id	Req. statement	Req. desc.	Comments	Justification of Requirement
		to the testbeds to expose one or the other type, or both.		
ST.1.015	Easy application development and deployment on Fed4FIRE	Fed4FIRE must provide the means for experimenters and third parties to develop and/or deploy applications on top of Fed4FIRE infrastructure. This should be done in such a way that these external providers find it attractive to eventually join the federation. An experimenter might be able to install own pieces of software on resources of the testbeds involved in his experiment. The experimenter will also have the possibility to manually complete the experiment data and to setup initial data-sets over different testbeds. An interface through which the experimenter can enter these data must be provided for this and the data entered must be stored in the selected testbed to be used at runtime.	This involves suppliers might become part of the core federation players	Experiments need starting conditions and input data ready so the experiment can begin running
ST.1.018	Experimenters should be able to manage resources they have reserved at testbeds	Experimenters should be able to manage resources they have created (share/unshare/destroy)	Experiment replicability might benefit from ability to publicly share VM images	

Measurement & Monitoring

Req. id	Req. statement	Req. desc.	Comments	Justification of Requirement
ST.2.007	Real-time aggregated monitoring management control	Fed4FIRE must provide tools to create, view, update, and terminate monitoring configurations related to shared resource types or experiments in real time. Monitoring data should be reportable for visualisation and analysis purposes with several reporting strategies (only alarms, all data, filters, etc.) in real time in order to provide accurate information and ease the analysis process. The experimenter might create own aggregated/composite monitored elements out of the available ones when designing the experiment, deciding what to monitor, defining some filtering possibilities as well, and providing the destination endpoint to send the information to. Monitoring information must cover information from different testbeds and services. Monitoring metrics should be compatible across different testbeds. For example, Monitoring computing & network resources' capacity. The monitored data during an experiment runtime will be available to the experimenter for all components involved in the experiment. If not aggregated, at least monitoring information from all involved testbeds must be provided. Testbeds must be able to publish infrastructure status through an API	As most of the monitoring in cycle 1 is done by using OML streams, experimenters should be able to create and/or configure new/existing OML Measurement Points in real time (maybe via FRCP)	Access to experiment monitoring should be defined by the experimenter and not by the testbeds themselves. This could be extended to infrastructure monitoring in the context of a experiment. Experimenters should be able to select the metrics they need to monitor, so they can get information that is useful and meaningful to them. Give the experimenters the basic monitoring metrics and the ability to aggregate them. Particular resources may need specific control & configuration interfaces to unburden an experimenter from the need to know about detailed testbed infrastructure capacities and architecture. Additionally, due to differences in the equipment a resource having identical functional features may have different control & configuration capacities. Experimenters need to be informed quickly of any notifications - e.g. breakdowns or

Req. id	Req. statement	Req. desc.	Comments	Justification of Requirement
				errors, so they can react to them

Trustworthiness

Req. id	Req. statement	Req. desc.	Comments	Justification of Requirement
ST.3.002	Single sign on with a single set of credentials for all testbeds and tools required for experiment	Fed4FIRE should enable the experimenter accessing all resources (hardware and software)/TESTBED with own privileges by performing login only once at the start of the experiment (single sign on). The task of setting up accounts and learning how to use the tools needed for the different stages of the experiment lifecycle (definition, deployment, execution, monitoring and gathering of results) only has to be performed once for the whole set of heterogeneous testbeds. Experimenters will use a single set of credentials for all operations on the platform (experimenting, monitoring, etc.) regardless of the tool used (handover). The authentication and authorisation processes to grant access to resources should be as light as possible for the experimenter. Finally, any Identity information must be protected within Fed4FIRE.	It should enable the creation of an account on different testbeds and/or services and link them to a unique Fed4FIRE user. ID management and mapping. Privacy and data protection are also important. For example, passwords must never be sent by email.	It is desirable to have a single sign on for all facilities - makes things easy for the experimenter. Having one user id for all testbeds, resources and tools eases accounting and audit to be managed more easily than if there are multiple ids. The protection of the identity is critically important, as failure to protect users' identity will result in severe reputation damage. On the other hand, services provided by external parties might require the experimenter is authenticated against this service provider in order to use the service (above all if the service provider does not belong to the federation).

Req. id	Req. statement	Req. desc.	Comments	Justification of Requirement
ST.3.009	Authorisation	Fed4FIRE will make the distinction between requests of local users, PhD students from other institutes (research), students (practical exercises), in order to know what kind of experimenter is logging in and from where and apply policies accordingly. Fed4FIRE must provide secure mechanisms to retrieve information produced by experimenters during experiments or to use resources privately. This access should be controlled according to these profiles. Moreover, Fed4FIRE must provide the means to authenticate different users belonging to several organisations that might collaborate in the same experiment. Of course, within the limits accepted by all parties in the federation, testbed owners should be able to control access to their testbed. Finally, Within the limits accepted by all parties in the federation, testbed owners should be able to grant credentials to experimenters outside the scope of the federation for local usage.	Identification of user roles – policies could define rights users may have on resources - e.g. PhD students have less rights than a principal investigator. If they have complementary expertise, which is usually the case, they might be accessing different testbeds (not the same) for the same experiment, but this should not be restricted.	There are real cases where different classes of user have different access rights and privileges, and this is practiced in the testbeds in the project. Infrastructure providers need to be able to establish access policies in order to prevent misuse. Controlled sharing of experiment data and results can be very useful for experimenters. Also, some experiments can be carried out by more than one institution.

Interconnection

Req. id	Req. statement	Req. desc.	Comments	Justification of Requirement
ST.4.003	Access between testbeds (interconnectivity)	WAN links/public internet access between testbeds with detailed monitoring for accountability (for testbed provider) so that during an experiment, experimenters can invoke services inside and outside a testbed, moving data between testbeds without it going through his personal machine. Running an experiment on resources provided by different hosting organisations. Information must be retrieved from several testbeds and processed in another one. Also, replicate data between different clouds and redirect requests to the appropriate data centre to retrieve the information.		This is an important requirement because it enables testbeds to talk to one another - this is a fundamental part of the federation.

Appendix C Requirements from SLA management

Methodology followed

The task of gathering requirements from all testbeds has a main objective: to understand what the SLA management should do in Fed4FIRE. For this, the participation and involvement of the testbeds is considered to be the most important factor since SLA management is usually associated to something beneficial for the customer and not for the providers. Still, providers (testbeds in our case) can indeed take advantage from SLA management, since it can provide interesting insights concerning self-performance and because they can be used to protect the testbeds.

This is why the process in Fed4FIRE has first focused on the testbeds. A round of interviews with every testbed has taken place consisting on:

- A brief presentation of:
  - Introduction of SLA management.
  - Benefits for the testbeds.
- A survey/interview filled by each testbed in order to know:
  - Their SLA management roadmap, if any.
  - Their expectations from Fed4FIRE SLA management (direct requirements).

Of course, the requirements gathered for cycle 1 have also been taken into consideration, since they were not tackled in the first iteration of the Fed4FIRE architecture.

The testbeds who have participated in order to obtain the SLA requirements are:

Testbed	Institution
Experimenta	i2cat
Netmode	NTUA
Planetlab Europe	UPMC
BonFIRE	Inria
NITOS	UTH
FuSeCo	Fokus
Koren	NIA
SmartSantander	UC
Ofelia	Ubristol
Norbit	NICTA
Virtual Wall	iMinds
w-iLab.t	iMinds

Most of these testbeds do not operate or do not intend to operate commercially. They all offer resources and/or services on a best effort basis and do not have SLA mechanisms currently in place.

As for proposals presented to the first Fed4FIRE Open Calls, they have not mentioned SLA requirements, most probably because the information provided to participants corresponds to cycle 1 achievements, among which SLAs are not included.

Benefits for the experimenters

It is obvious that SLA management is beneficial for the experimenters since the SLA helps them know what to expect and because the SLA gathers all the terms of the service and thus can be helpful in order to resolve disputes. For the experimenter, the SLA is a guarantee for the service offered.

Benefits for the testbeds

The testbeds (and service providers in general) can also get benefits from SLA management, even if they do not operate commercially:

- Commercial operation requires quantification of services in order for customers to measure what they are paying for (response time, bandwidth, throughput ...)
- Quantification of the service a testbed provides helps experimenters know exactly what they are being offered by a testbed, which is important in case of later charging/billing or just for reputation reasons.
- SLAs can also be used for a better management of resources and services. For example, a testbed can get information concerning its capacity and how well it is performing. A testbed may find its optimal operational capacity (not too much resources free –meaning no wasting of resources–; not too much resources occupied –more users could be reached–).
- As a testbed approaches full capacity, the operation is more efficient, but resource management becomes more challenging. For example, SLAs can help decide which requests should be accepted and which ones rejected when the testbed receives more requests than it can handle, or help decide when to purchase more resources.
- SLAs allow offering different levels of service at different pricing levels, e.g.:
  - Cheap best effort service, where the user may get failures at any time
  - Top quality guaranteed service, which is more costly but the user gets the resources they need exactly when they want them.
- SLAs can also be used to control customers and protect testbeds by avoiding abusive usage and fraud (e.g. logging in with a fake profile) as expressed in the terms and conditions.

The SLA management information goes beyond monitoring information; it is more elaborated and it contains insights concerning the testbed behavior that can be used by testbeds for many different purposes (e.g. reputation).

Please note that SLA management does not provide extended monitoring to testbeds. SLA management takes available monitoring information as an input and compares this with the SLA, in order to evaluate if the latter is being met or not. This implies that the **SLA management relies on and is limited by the monitoring information provided by each testbed**. Any enhancement required for specific SLA management at every testbed will have to be put in place in the framework of WP6.

Besides, although SLAs may contain pricing information for a certain service or resource usage as one of the terms and conditions, the SLA management does not implement billing features. Instead, depending on the result of the SLA evaluation –which is done as part of the SLA management-, the system could send some information to some billing system used by a testbed, either for credits or charges to be added to the current bill in case it is required.

Finally, for testbeds willing to adopt SLAs but not offering any special commitments beyond best (or reasonable) effort, this tool can still be helpful in order to increase reputation or to provide information about own behavior. In the case of best effort, the result of the SLA evaluation is always

"fulfilled" whatever performance presented by the provider, but in case testbeds are able to provide at least some commitments which do not need to be fine grained (e.g. my testbed is up and running 95% of the time), some SLA work could be done on them.

Requirements

This subsection gathers all SLA requirements collected during the requirement elicitation phase. All requirements are presented using the same format:

Source	Req. id	Req. statement	Req. description	Comments	Justification of Req.	It affects

- Source: The testbed that has provided the requirement.
  When the requirement does not come from any particular testbed but from cycle 1 it is considered as generic -GEN.
- Req. Id: Requirement Identifier to ease tracing ("ST.Area.number")
  - ST stands for Services and Tools
  - Areas :
    - 1: Experiment Workflow and Lifecycle Management
    - 2: Measurement and Monitoring
    - 3: Trustworthiness
    - 4: Facility Management
  - Requirement number: 001, 002, etc.
- Req. statement: Brief description of the requirement.
- Req. description: Descriptive text for the requirement.
- Comments: additional information regarding the requirement
- Justification of Req.: Provides the reason why requirements have been included (the aim of the each requirement).
- It affects: This column contains implications for other areas of Fed4FIRE.

Source	Req. id	Req. statement	Req. description	Comments	Justification of Req.	It affects
Koren Experimenta	ST.3.001	Profile-based offering	Different SLAs will be offered depending on the experimenter's profile: e.g. academia will be offered Best Effort and industry will have Premium services offered. More or less resources are provisioned according to the SLA.		Koren will offer BW guarantees between datacentres. Experimenters sign up for the specific BW (intervals of Mbps). Profile -based offerings: best effort free for academia and Premium services for industry (providing more resources to customers in exchange of a fee (experiment-based mostly). In case the SLA is not met, they can repeat their experiment.	Authentication and profiles belong to the security domain.
Experimenta	ST.3.002	Prioritisation of requests	There will a prioritisation of some users' requests over others (premium over normal). Moreover, resources will be assigned or reserved for a user until the experiment finishes, unless he makes a use against the testbed policies.		SLAs can be used for managing requests according to the capacity and the demand. This demand may come from users with different profiles, which might help the decision	Authentication and profiles belong to the security domain. The user profile is present in this process so that the testbed can accept or reject the request
Koren	ST.3.003	Deeper insight of testbed usage	The SLA management system should help testbeds retrieve intelligent information concerning the testbed usage (e.g. track historical usage per experimenter)		Koren wishes to have more control over the datacentre (Point-of-Presence). Keep track of historical usage, user registration, in general, a deeper insight of usage is requested.	Monitoring. User information should be provided to the SLA management
GEN	ST.3.004	SLA template specification	Fed4FIRE must provide a clear step-by-step procedure describing how to write and SLA template for infrastructure providers		Each testbed provider needs to describe their offerings, commitments and terms of usage in order to expose them to experimenters	
BonFIRE Virtual Wall w-iLab.t	ST.3.005	SLA establishment	The SLA system should not be bureaucratic, it should not add additional administrative burden for experimenters. Simple, costless and short SLA establishment.		The biggest complaint users have from publicly funded initiatives is bureaucracy	

Source	Req. id	Req. statement	Req. description	Comments	Justification of Req.	It affects
BonFIRE, FuSeCo	ST.3.006	SLA for reservation in exchange of quota	For reservation-based testbeds, a reservation is a time slots on resources. The SLA management should help keep track of the booking diary, by providing the information of whether reserved resources have been actually used or not and raise alarms in case of conflict. Tentative reservations could be offered at a lower price (price in terms of quota). Besides, this should work even if the reservation system is vague “2 weeks sometime in the next month”.	BonFIRE will offer reasonable effort (not Best since "best" is not specific) Reservation-based SLA and the “money” is the quota (CPU, storage). When the user has reserved, he is prior, no one will use the resources as long as there is no force majeure issue. If there is a crash, their quota is given back to them. As for operational organisation, BonFIRE can guarantee not making maintenance in reservation slots. Otherwise a compensation with additional quota will be done.	This allows the possibility for a sort of penalty applied to experimenters in case of misuse of the reservation system once they have committed.	This requirement implies that the SLA management will have to indicate to BonFIRE that there is an adjustment to make on the quota. Moreover, additional monitoring in Fed4FIRE needs to be done: the info concerning the reservations, the cancellations, how much time in advance reservations are made, etc. is in the dB but there is no current process to interpret it.
Norbit Planetlab Netmode, BonFIRE, FuSeCo	ST.3.007	SLA for reservation and Best Effort for availability	The SLA management should be aware of reservations for those testbeds who operate on this basis. This means that there will be a previous phase before the experiment in which the reservation will take place and an according SLA has to be setup.		Several testbeds provide a resource reservation mechanism so experimenters can book a set of nodes for a specific period of time. The reservation system guarantees the assignment of nodes. However, the system will work on a best effort basis regarding the nodes availability. The availability of the nodes during the reserved period of time is not guaranteed as communication issues, power outages or node malfunctioning, for example, might turn nodes unreachable. BonFIRE, Planetlab, NORBIT, NETMODE, FUSECO are reservation-based testbeds (although reservation is currently made manually using a calendar).	The SLA management must be integrated with reservation mechanisms

Source	Req. id	Req. statement	Req. description	Comments	Justification of Req.	It affects
Experimenta	ST.3.008	SLA offerings and pricing	Possibility of offering Best Effort and other Premium alternatives in exchange of a fee		Experimenta can offer different offerings: Best effort: minimum commitment, data custody, experiment permanent storage (configuration and topologies, VM requested), service availability and maintenance, max. number of running experiments, probably restricted access. Limited number of resources. Premium: Priority (access priority), all resources available different user types, users paying a fee (premium usage). Better response time. Data custody would be equal for all but access wouldn't	Extended monitoring is required for this. Conciliation is beyond the scope of SLA management.
GEN	ST.3.009	Accountability	When an experiment goes wrong (time out, etc.), it is essential to distinguish whether the problem is originated in the infrastructure (SLA not met) or if this is caused by the experimenter's own software or data-set used during the experiment.		This is important in order for every party to claim their share according to their responsibilities	
GEN	ST.3.010	SLA publication and discovery	Fed4FIRE must provide the means for infrastructure providers to publish offerings, experimenter requirements in terms of QoS and browse/compare offerings. User must be aware of available offerings (including costs associated with resources provided) using specific criteria (e.g. privacy and cost).		This is the most dynamic, sustainable and efficient way of bringing infrastructure providers and experiments closer	

Source	Req. id	Req. statement	Req. description	Comments	Justification of Req.	It affects
GEN	ST.3.011	SLA monitoring	Ocne the user begins the experiment, the SLA management will be able to compare each individual SLA with the monitoring of every testbed involved.		The SLA management takes available monitoring information as input and evaluates if the agreed SLA is being met	The SLA evaluation depends on the information provided by the monitoring system for every testbed. The SLA tool needs to be aware of the start and end times of the experiment
GEN	ST.3.012	Unmet SLA by providers	In case the SLA is not met, the SLA management system must be able to clearly identify and make visible which testbed(s) did and did not commit. In case an SLA is not met, then the actions defined in the SLA for the testbed that did not meet the SLA must be invoked.		All testbed providers agree that this requirement is important for trust, since there will be best effort testbeds used along with premium testbeds with different terms and conditions within the same experiment.	For this, the SLA management will require to be interfaced with external systems that will actually carry out these actions. This applies also for reservation.
GEN	ST.3.013	Unmet SLA by experimenter	In case experimenters do not meet their obligations, the SLA management must facilitate the claim for some compensation by all affected testbeds who require so.		This is an important requirement for testbed providers. SLA management can help them protect their testbeds.	
Smart Santander FuSeCo	ST.3.014	SLA evaluation for individual testbeds. SLA management, monitoring and capacity planning	The SLA management will allow knowing how much capacity is required for the SLAs offered and what SLAs a testbed can offer with the current available capacity.		For each SLA offered, the testbed must know how much capacity is required and which SLAs could be met	

Source	Req. id	Req. statement	Req. description	Comments	Justification of Req.	It affects
GEN Ofelia	ST.3.015	SLA negotiation based on offerings	There will be a mechanism for a negotiation to agree SLA conditions between the experimenter and each provider.		Infrastructure providers could offer a limited number of offerings (e.g. Best Effort and Premium) at different prices and experimenters would choose from a menu	
GEN	ST.3.016	Experimenter's obligations	The SLAs must gather, for each testbed, the obligations of the experimenter. For example, the user must be associated to a site (e.g. academic institution in the particular case of Planetlab.)			
GEN	ST.3.017	SLA violation notification	Users will be informed of SLA violations		Experimenters would appreciate to be informed of SLA issues	
GEN	ST.3.018	Aggregate SLAs and monitoring	Fed4FIRE must provide the means to manage and monitor an aggregated SLA lifecycle by joining partial SLAs provided by the underlying infrastructures and services. Monitoring system values related to the SLAs is required.	This requirement requires ST.3.010 to be implemented. Aggregated SLAs will be built joining individual ones.	A global SLA is a federation-level SLA managed by Fed4FIRE	
GEN	ST.3.019	SLA establishment	Fed4FIRE will allow the establishment of an SLA between the experimenter and testbed providers. Before the experiment begins, all related SLAs must have been signed (including reservation SLAs).		Important because SLAs are key to enable management of resources by the testbed provider and management of expectations and cost by the users.	

Source	Req. id	Req. statement	Req. description	Comments	Justification of Req.	It affects
Smart Santander	ST.3.020	Operational planning	In case an experiment involves several testbeds and at least one of them is reservation-based, Fed4FIRE should ensure that none of the involved testbeds plans maintenance operations that could affect the experiment during reservation time		This is a federated service	
Ofelia FuSeCo	ST.3.021	Offering based on capacity	The SLA mechanism should provide the means for the testbed to expose a certain SLA according to the available capacity at a given moment.		We need to know when resources are approaching limits that will prevent them from being able to deliver the quality of service, and what to do about this situation. For example, if only a portion of the total nodes are available, the testbed provider can offer a suitable SLA, adapted to its current capacity.	Monitoring. This information should be provided to the SLA management
Ofelia	ST.3.022	Integration with existing tools	The SLA mechanism must adapt to the different policy engines existing in different testbeds, if any. If a central policy component is included in Fed4FIRE, the SLA will interact with it.		OFELIA has a policy engine that is triggered every time an experiment is made. The policy engine determines when a request is rejected according to capacity rules. The SLA framework of Fed4FIRE must integrate with this tool.	
Ofelia	ST.3.023	SLA signing entity (on behalf of the experimenter)	The SLA mechanism in Fed4FIRE must contemplate the fact that experimenters can follow hierarchies as far as registered users are concerned (e.g. institutions, projects, research fellows) and that, in some cases, the signature of a high-level user is valid for all underlying users. This policy might be different for every testbed.		The SLA management should try to avoid multiple signatures by users belonging to the same "experiment domain"	This is related to authentication mechanisms. There should probably be an SLA responsible in every user group, understandable by Fed4FIRE

Source	Req. id	Req. statement	Req. description	Comments	Justification of Req.	It affects
GEN	ST.3.024	General terms and conditions	Fed4FIRE must provide the means for every experimenter and testbed provider to be aware and sign the terms and conditions of the federation		As described in the Fed4FIRE CA, experimenters and testbed providers become partners and must agree with the terms and conditions before joining the federation	
FuSeCo	ST.3.025	SLA Management	SLA management should provide machine readable information in order for the monitoring to know what parameters to measure in order to fulfil the SLA and anticipate upcoming failures. This information must be automatically provided before provision is achieved.		This allows testbeds to operate more efficiently.	
Virtual Wall	ST.3.026	SLAs for tools provided by software suppliers	Fed4FIRE will provide the means to setup SLAs also for applications developed by third party suppliers provided that they become part of the federation		For a supplier to be controlled, it is a prerequisite that it can be identified as a player within FEd4FIRE; for which the provider will have joined before. Otherwise, no SLA management will be possible.	The central architecture of Fed4FIRE, which must enable this new kind of players. It also affects the Fed4FIRE sustainability definition as these stakeholders will become core players once they federate.
GEN	ST.3.027	Reputation	The SLA management will provide information to the central reputation component in Fed4FIRE		SLAs affect reputation and this a valuable input to this component	The central reputation component
GEN	ST.3.028	Different kinds of services	In Fed4FIRE, there are low level resource services (e.g. testbeds offer raw resources) but also high-level application services (that run over the resources). Fed4FIRE must be able to deal with SLAs for both kinds of services.		This can be seen as different ranges of services, from low level to high level ones	The central architecture will enable both types of services

Source	Req. id	Req. statement	Req. description	Comments	Justification of Req.	It affects
Virtual Wall	ST.3.029	Openness and Reusability	The SLA management in Fed4FIRE should be open and based on market standards		The strength of the federation lies in defining standards and frameworks of which everyone is free to adopt or not, but by adopting them, you have real advantages. In Fed4FIRE, federation is a very loose coupling of parties, but a natural coupling by using the same APIs and conventions	The testbeds and the central architecture

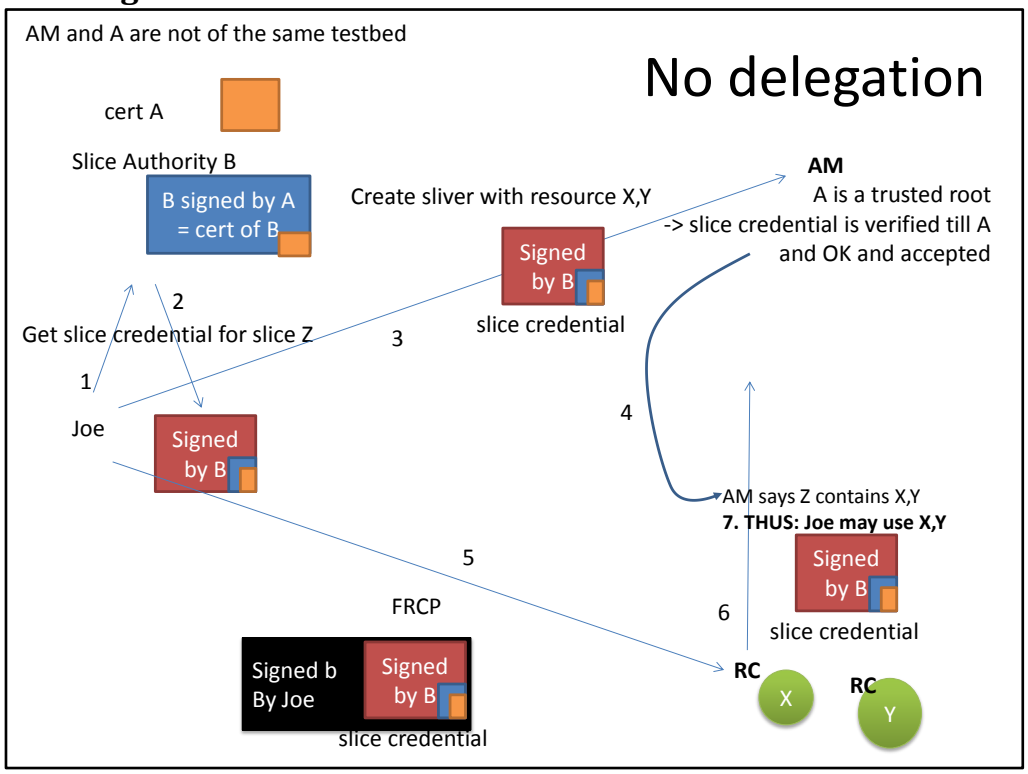
Appendix D: Interaction between Policy Decision Point and Aggregate Manager

Simply put, a slice is a container owned by a slice owner and issued by a Slice Authority - think of it like a shopping bag that can be filled with resources from multiple testbeds. The Slice Owner can go to a testbed's AM and request resources in that slice. If the AM agrees, it will allocate resources for the slice owner and bind them internally somehow to the slice. After this, the slice owner can go to another testbed and request more resources in the same slice. The slice then contains two sets of resources at two testbeds, and the associated slice credential is a way of proving the ownership or rights on the slice, and therefore to the resources at both testbeds allocated to the slice.

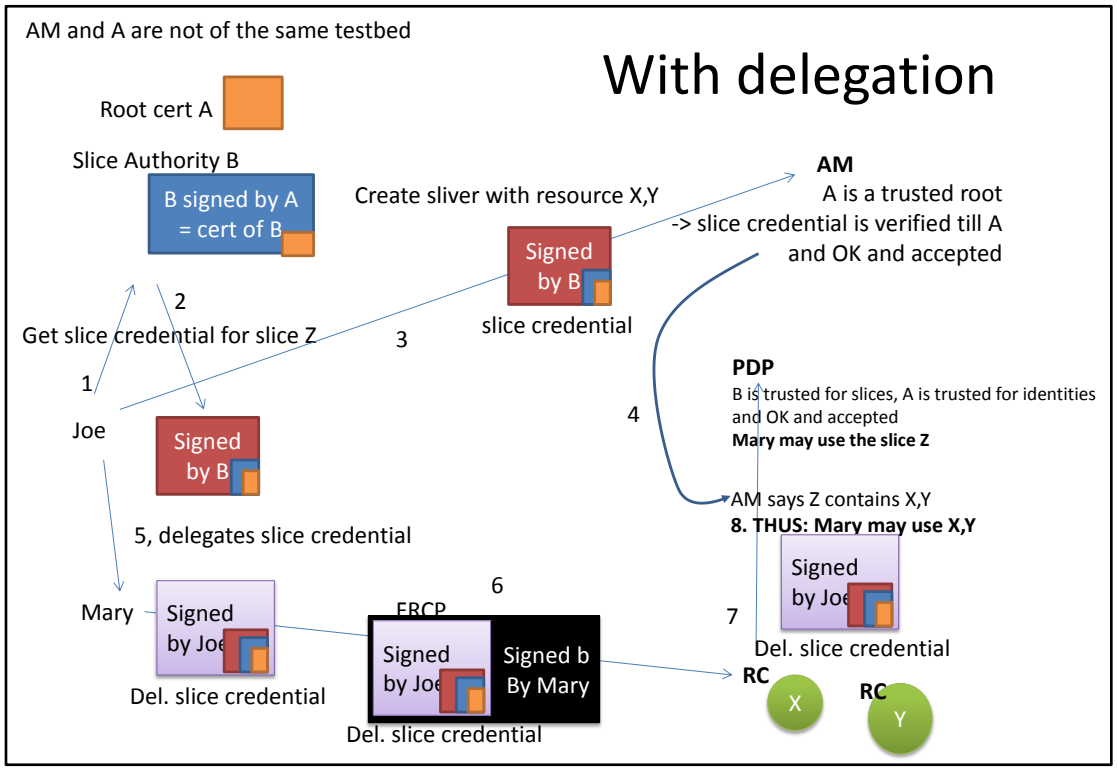
The PDP uses the SFA slice credential as a common access token in multiple testbeds. The PDP installed at all testbeds who want it, and it protects an OMF RC at the testbed - this RC is the point of entry for users of resources. This approach (and the PDP) is not connected with SSH keys at all - the PDP is another way of accessing resources reserved in SFA and supports the access via FRCP and OMF. Testbeds can support SSH access, FRCP access, or both - it is up to them which ones they support, but SSH and FRCP access to resources are completely separate. The PDP is intended to bridge resource request and allocation using SFA, and resource usage in FRCP. The PDP provides a way for an AM to declare the binding of resources at a testbed to a slice, so when the slice owner presents a slice credential, the PDP will be able to decide whether they can access the resources.

There is no need for a common account in this pattern, because the slice is the common mechanism by which resources at multiple testbeds can be requested and used, and the access credential for this is the slice credential. However, when the owner of a slice wants to provide access ot its resources to other experimenters also, some additional measures need to be taken: As depicted below, these can be based on different approaches: without delegation of the slice credential, or with delegation. The corresponding steps are depicted in the figures. More details will be given in the context of WP7.

No delegation



With delegation of slice credential



Appendix E: What information does a user credential and slice credential contain

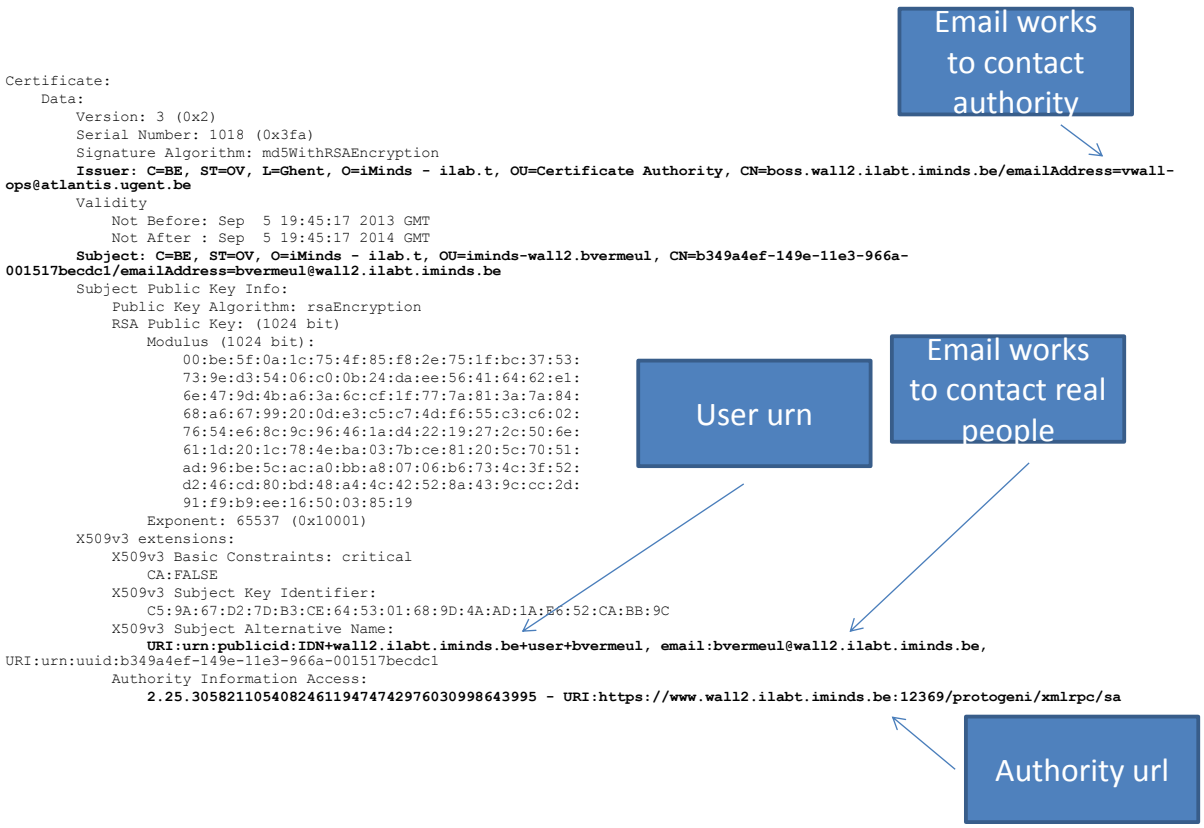
Slice credential

A slice credential is an X.509 certificate, and is therefore intended to contain information about the slice in a way that any recipient of the credential can authenticate it. In other words, it is a document that allows the transfer of information about a slice in a trusted manner. As depicted below, the slice credential contains information about the (sub)authority that issued the slide, the validity period of the slice, the slice name and the slice owner. For further details we refer to WP7.

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 3621 (0xe25)
        Signature Algorithm: md5WithRSAEncryption
        Issuer: C=BE, ST=OV, L=Ghent, O=iMinds - ilab.t, OU=Certificate Authority, CN=boss.wall2.ilabt.iminds.be/emailAddress=vwall-
ops@atlantis.ugent.be
        Validity
            Not Before: Nov 11 12:24:44 2013 GMT
            Not After : May  4 13:24:44 2019 GMT
        Subject: C=BE, ST=OV, O=iMinds - ilab.t, OU=iminds-wall2.pdptest2, CN=a12e2ec1-4ad4-11e3-883a-
001517becdc1/emailAddress=brecht.vermeulen@iminds.be
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            RSA Public Key: (1024 bit)
                Modulus (1024 bit):
                    00:dd:28:80:bf:b3:27:6a:59:f9:ca:e9:89:d5:12:
                    51:aa:db:84:15:fa:33:f7:ba:21:68:71:98:71:b5:
                    e5:ad:73:41:f6:f2:99:fa:7a:6a:0b:f2:db:d7:4f:
                    db:83:b8:8e:57:7b:d3:50:ea:d5:18:df:29:5e:f8:
                    8b:46:37:c0:a3:e4:4d:2e:4e:67:6e:fa:6f:57:23:
                    a7:ee:85:08:7e:9b:3a:25:8b:c8:ea:a0:96:4a:68:
                    48:cf:73:c5:a0:ab:f6:b1:fa:38:79:65:5f:ab:1d:
                    f1:70:bb:6b:e4:3f:df:27:dc:9e:45:5f:1d:5e:47:
                    db:20:22:de:68:61:4a:f3:2f
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Subject Key Identifier:
                42:36:EA:B6:6B:D5:C0:44:88:33:D1:2D:2E:98:47:32:F3:1F:BF:76
            X509v3 Subject Alternative Name:
                URI:urn:publicid:IDN+wall2.ilabt.iminds.be+slice+pdptest2, URI:urn:uuid:a12e2ec1-4ad4-11e3-883a-001517becdc1
            X509v3 Basic Constraints: critical
                CA:FALSE
```

User credential

A user credential is an X.509 certificate, and is therefore intended to contain information about the user in a way that any recipient of the credential can authenticate it. In other words, it is a document that allows the transfer of information about a user in a trusted manner. Below more details are given about the information that is contained in the user credential that is returned by the member authority of the Virtual Wall. As depicted below, that user credential contains information about the (sub)authority that issued the user credential, the validity period of the credential, the user's contact information and the user's public key. Note that this presence of valid contact information is critical when accountability is to be supported. For further details we refer to WP7.




```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 1018 (0x3fa)
        Signature Algorithm: md5WithRSAEncryption
        Issuer: C=BE, ST=OV, L=Ghent, O=iMinds - ilab.t, OU=Certificate Authority, CN=boss.wall2.ilabt.iminds.be/emailAddress=vwall-
ops@atlantis.ugent.be
        Validity
            Not Before: Sep  5 19:45:17 2013 GMT
            Not After : Sep  5 19:45:17 2014 GMT
        Subject: C=BE, ST=OV, O=iMinds - ilab.t, OU=iminds-wall2.bvermeul, CN=b349a4ef-149e-11e3-966a-
001517becdc1/emailAddress=bvermeul@wall2.ilabt.iminds.be
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            RSA Public Key: (1024 bit)
                Modulus (1024 bit):
                    00:be:5f:0a:1c:75:4f:85:f8:2e:75:1f:bc:37:53:
                    73:9e:d3:54:06:c0:0b:24:da:ee:56:41:64:62:e1:
                    6e:47:9d:4b:a6:3a:6c:cf:1f:77:7a:81:3a:7a:84:
                    68:a6:67:99:20:0d:e3:c5:c7:4d:f6:55:c3:c6:02:
                    76:54:e6:8c:9c:96:46:1a:d4:22:19:27:2c:50:6e:
                    61:1d:20:1c:78:4e:ba:03:7b:ce:81:20:5c:70:51:
                    ad:96:be:5c:ac:a0:bb:a8:07:06:b6:73:4c:3f:52:
                    d2:46:cd:80:bd:48:a4:4c:42:52:8a:43:9c:cc:2d:
                    91:f9:b9:ee:16:50:03:85:19
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Basic Constraints: critical
                CA:FALSE
            X509v3 Subject Key Identifier:
                C5:9A:67:D2:7D:B3:CE:64:53:01:68:9D:4A:AD:1A:B6:52:CA:BB:9C
            X509v3 Subject Alternative Name:
                URI:urn:publicid:IDN+wall2.ilabt.iminds.be+user+bvermeul, email:bvermeul@wall2.ilabt.iminds.be,
URI:urn:uuid:b349a4ef-149e-11e3-966a-001517becdc1
            Authority Information Access:
                2.25.305821105408246119474742976030998643995 - URI:https://www.wall2.ilabt.iminds.be:12369/protogeni/xmlrpc/sa
```

Appendix F: Subauthorities in a member and slice authority

Basic idea

At least the emulab slice authority and the GENI GPO slice authority do support sub-authorities to make authorities and experimenters further scalable.
The idea is based on putting projects/sub-authorities in place, and experimenters can belong to one or more projects. Each project has one or more PIs or project responsible which can approve other members of that project.

Scalability

The scalability comes in two ways:

- Authorization can be based on the sub-authority, this means that a particular testbed (aggregate manager) can let in all members of a specific project/sub-authority, but not other experimenters of that authority.
- Project creation has to be approved by authority administrators, but approval for members in the project is done by the project PI (delegation)

Practical example

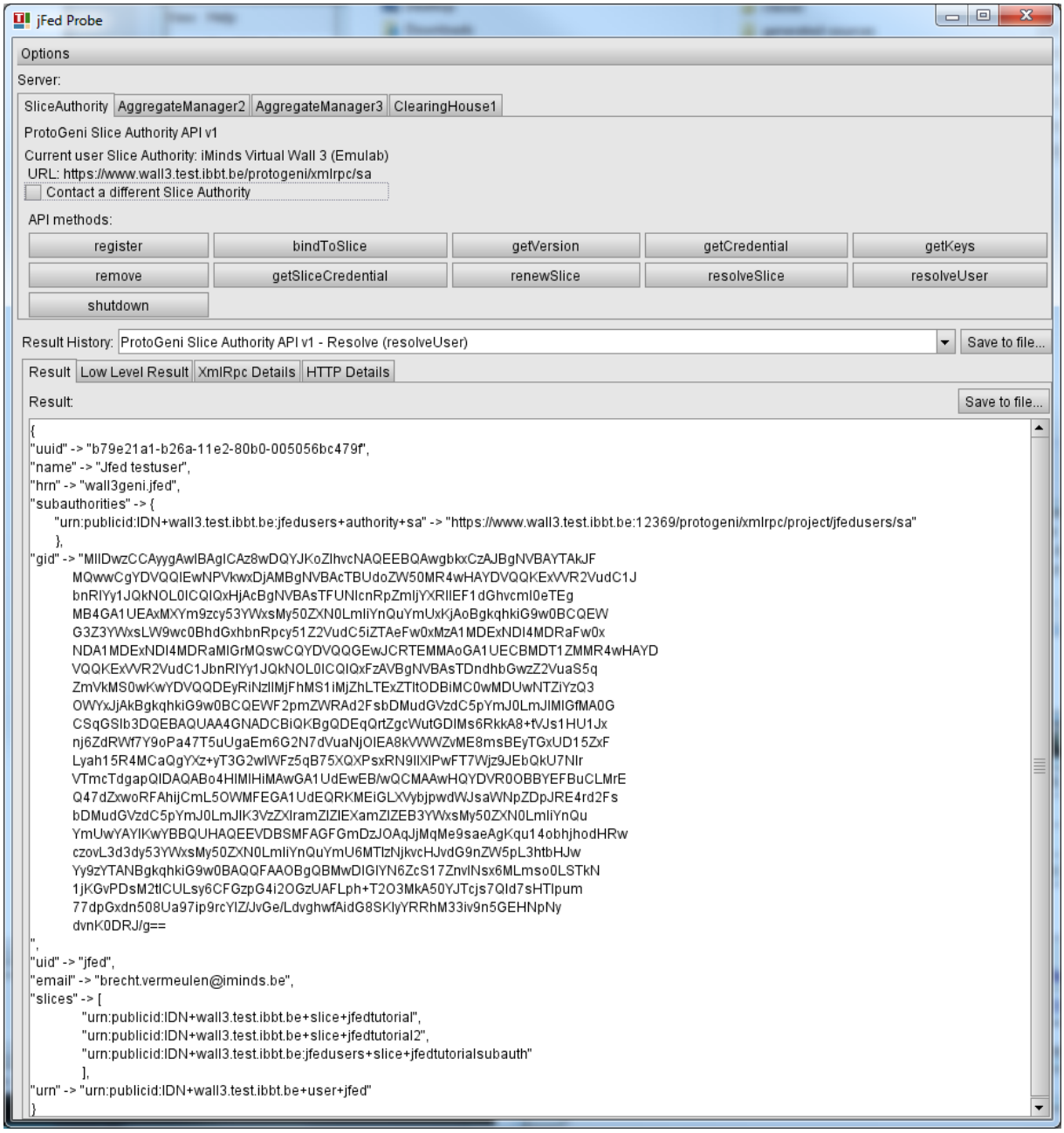
A practical example may make it clearer:

- iMinds' Virtual Wall has a lot of users: iMinds people, students, industry members, Fed4fire partners, …
- Without sub-authorities, if a user comes at another AM with a virtual wall certificate, authorization has to be done on the individual member (e.g. with attributes or based on email)
- With sub-authorities, the following can be done: a project 'networks4ever' (e.g. a European project) is created at the virtual wall, and contains a number of members. This project has agreed with Bonfire and Nitos that the members of the project can use the resources of those testbeds. At that moment, Bonfire and Nitos authorize incoming sliver requests based on the sub-authority 'networks4ever' signing the slice credential. So, if new members join the project, or leave the project, the 'networks4ever' PI can just add or drop them in the sub-authority, while Nitos and Bonfire just check the sub-authority (no matter if the project contains students, industry, …). All resource usage can then also be done in the responsibility of 'networks4ever' e.g.
- In the emulab authority it is not yet forbidden, but will be in the future, but in the GENI GPO authority, you can get a credential, but if you do not belong to a project/sub-authority, you cannot create slices. (and projects/sub-authorities can have a limited life-time, e.g. during a real project, or during a practical exercise for students or demonstration). E.g. the tutorials at GEC use projects which are limited to one week after the conference, so you can test at home, but only for one week under the tutorial authorization.

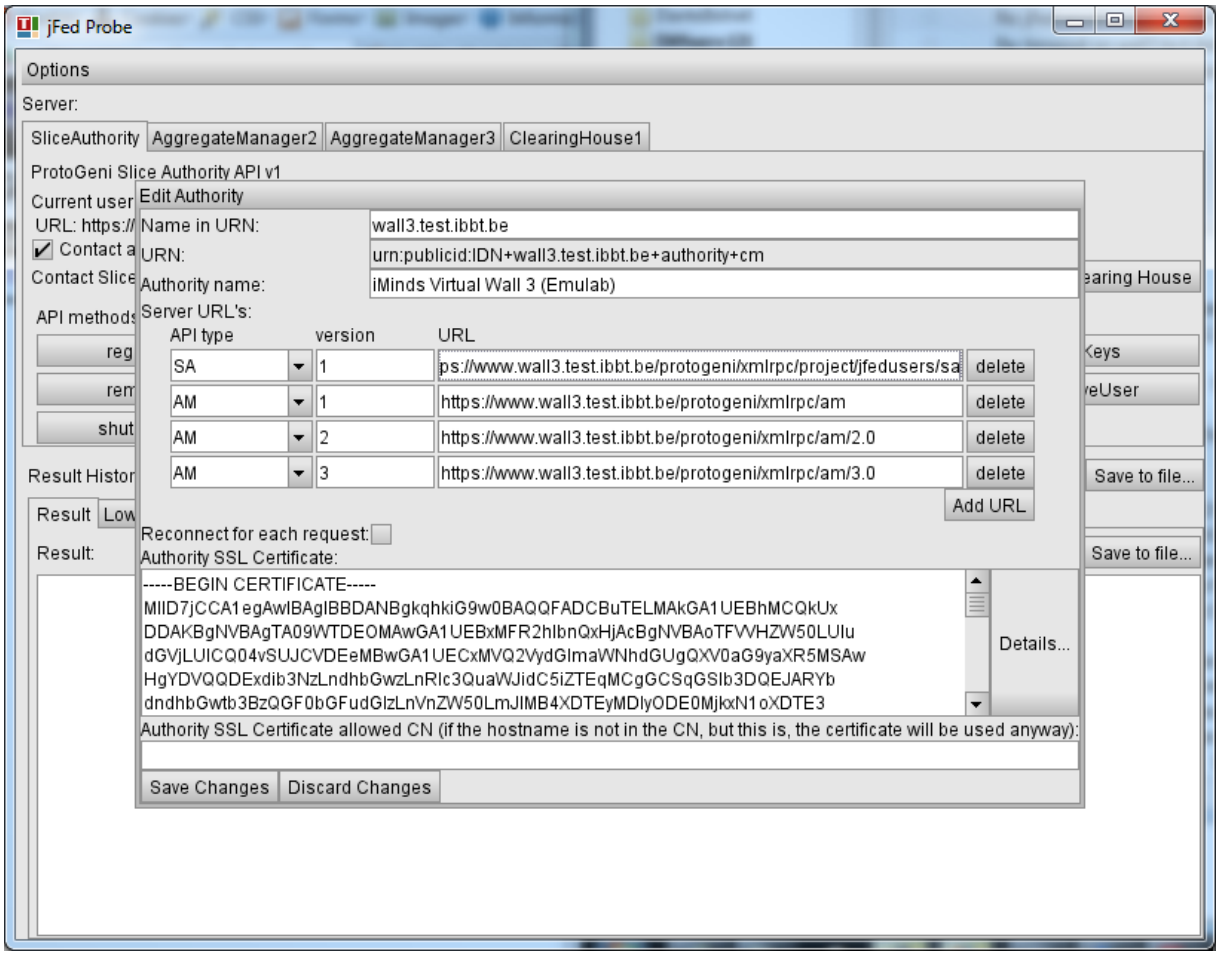
How to do this with the APIs by using jFed ?

At the end of cycle 1 the usage of subauthorities has already been explored and documented in the jFed tutorial (http://jfed.iminds.be) Since it describes how subauthorities can be used in real life in detail, it is a very useful additional illustration of the concepts discussed in this appendix. Therefore the most important parts of that tutorial have been copied below. Note that this tutorial is intended as an illustration of these architectural concepts, to make them as clear as possible. Such information can be situated somewhere between architectural descriptions and more detailed specifications. Therefore it is likely that this information (possibly in more detail) will also become part of WP5's and WP7's specifications for cycle 2 in deliverables D5.2 and D7.2. The remainder of this appendix is the extract of that jFed tutorial:

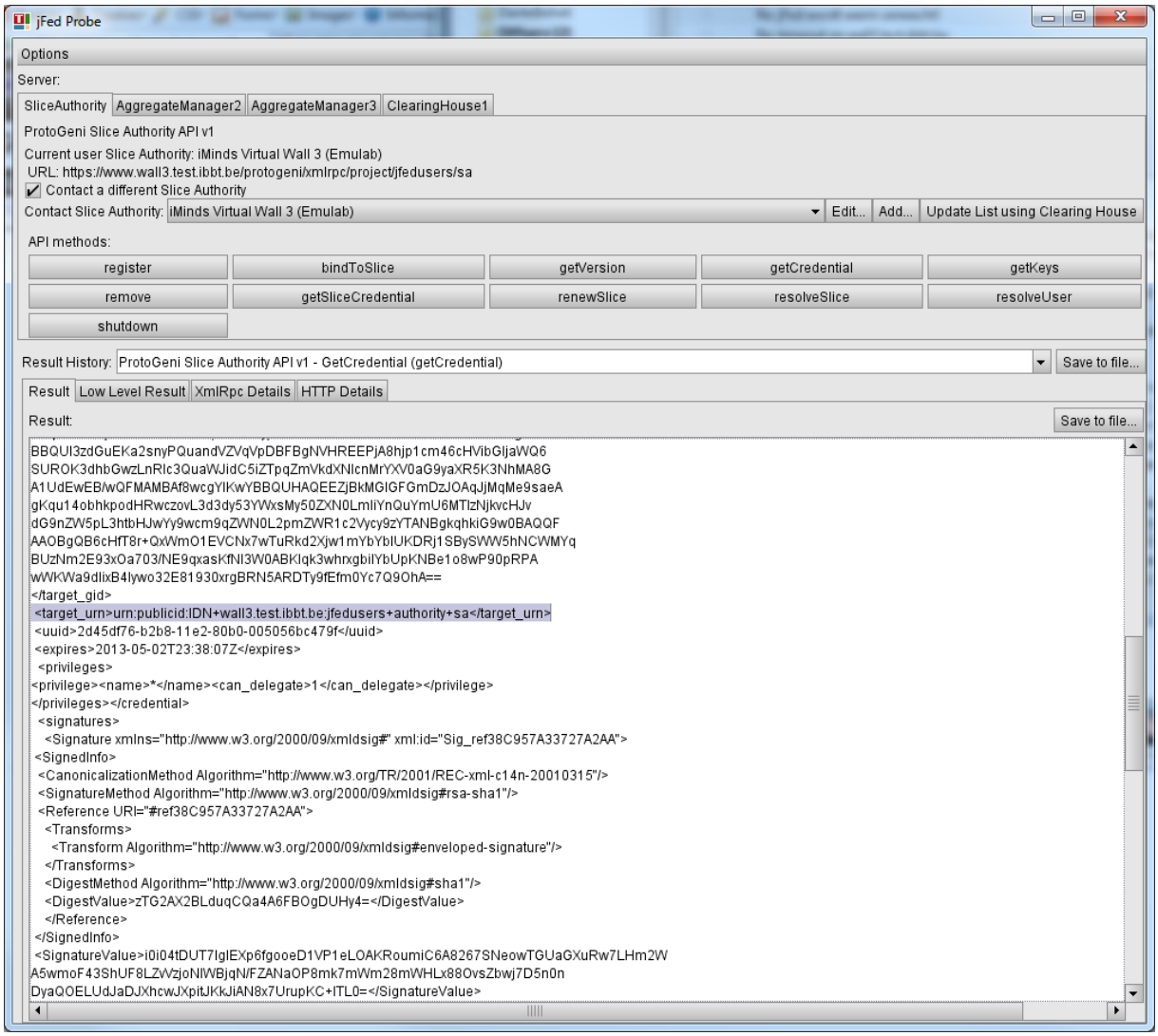
To demonstrate the concept of sub-authorities. A user can belong in multiple projects/sub-authorities. E.g. if you call 'resolveUser', you can see the subautorities to which a user belongs:



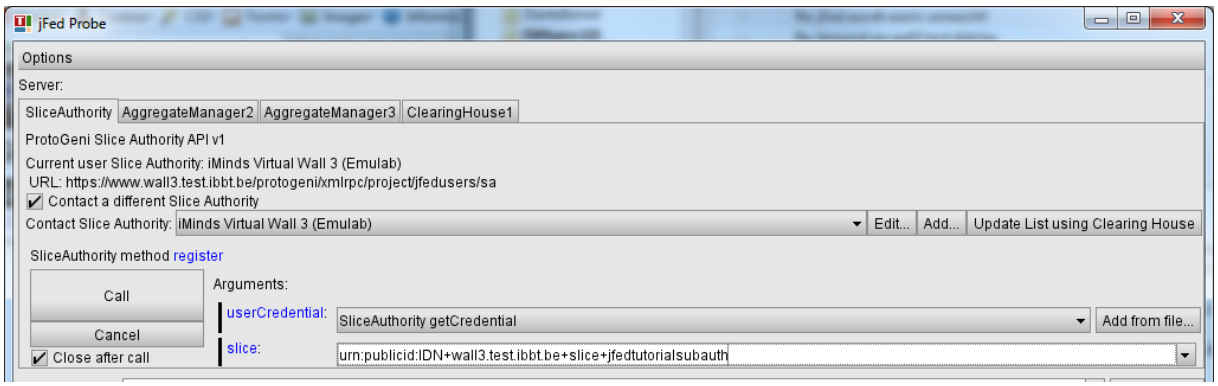
If you edit the slice authority, and put for the SA, a line as seen in the resolveUser (but without port 12369): https://www.wall3.test.ibbt.be/protogeni/xmlrpc/project/jfedusers/sa, you can create slices on this subauthority.
The advantage of this is, that other federating authorities can give access to a subpart of an authority, e.g. only slices belonging to jfed users are authorized.



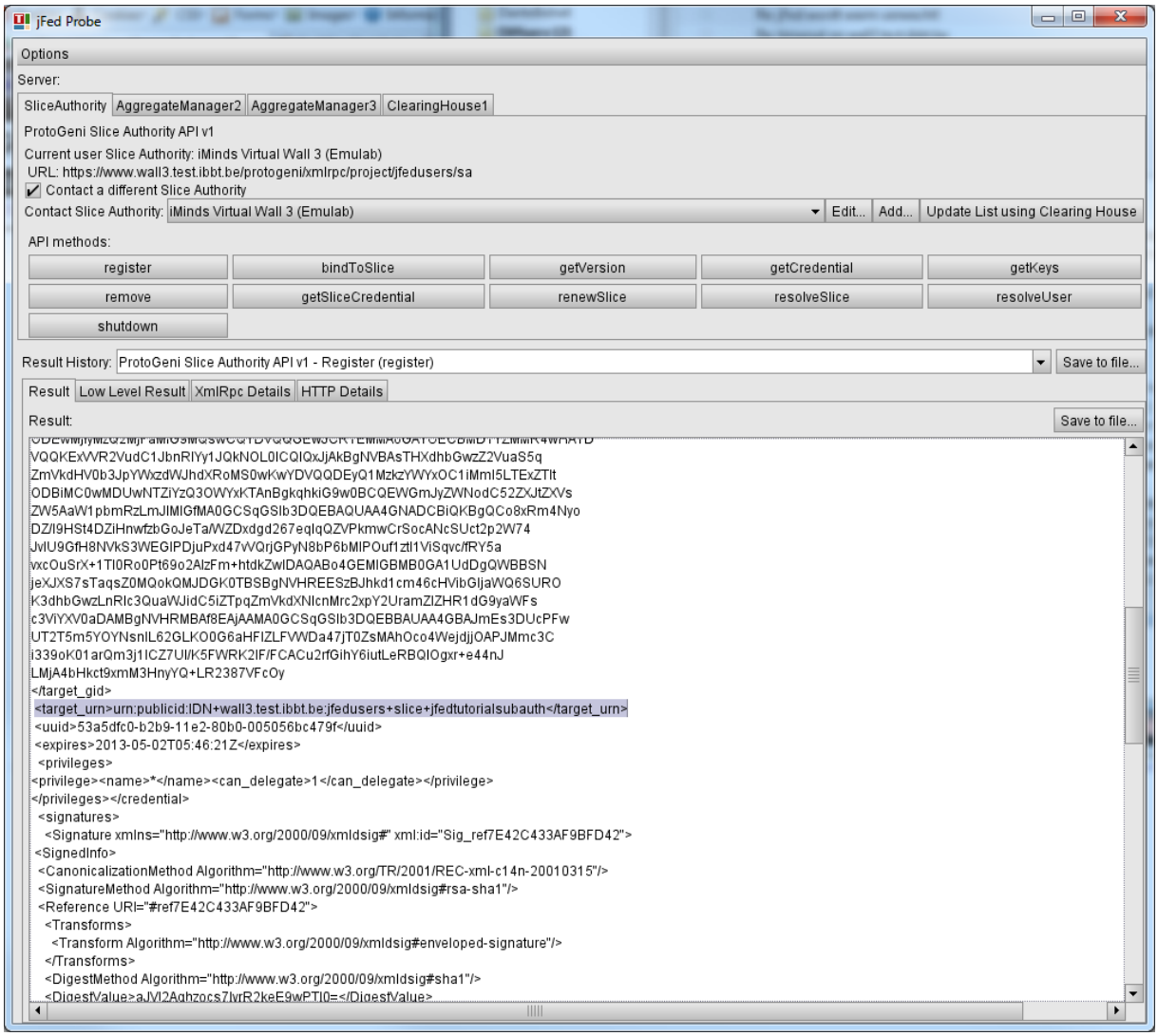
First, do a getCredential on this subauthority:



Then register a slice:



And you receive a slice from the subautority, which has a URN: urn:publicid:IDN+wall3.test.ibbt.be**:jfedusers**+slice+jfedtutorial



The details of this slice credential look like this:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<signed-credential                          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://www.protogeni.net/resources/credential/credential.xsd"
xsi:schemaLocation="http://www.protogeni.net/resources/credential/ext/policy/1
http://www.protogeni.net/resources/credential/ext/policy/1/policy.xsd">
  <credential xml:id="refB39085D4FD641534">
 <type>privilege</type>
 <serial>2807</serial>
 <owner_gid>MIIDwzCCAyygAwIBAgICAz8wDQYJKoZIhvcNAQEEBQAwgbkxCzAJBgNVBAYTAkJF
MQwwCgYDVQQIEwNPVkwxDjAMBgNVBAcTBUdoZW50MR4wHAYDVQQKExVVR2VudC1J
bnRlYy1JQkNOL0lCQlQxHjAcBgNVBAsTFUNlcnRpZmljYXRlIEF1dGhvcml0eTEg
MB4GA1UEAxMXYm9zcy53YWxsMy50ZXN0LmliYnQuYmUxKjAoBgkqhkiG9w0BCQEW
G3Z3YWxsLW9wc0BhdGxhbnRpcy51Z2VudC5iZTAeFw0xMzA1MDExNDI4MDRaFw0x
NDA1MDExNDI4MDRaMIGrMQswCQYDVQQGEwJCRTEMMAoGA1UECBMDT1ZMMR4wHAYD
VQQKExVVR2VudC1JbnRlYy1JQkNOL0lCQlQxFzAVBgNVBAsTDndhbGwzZ2VuaS5q
ZmVkMS0wKwYDVQQDEyRiNzllMjFhMS1iMjZhLTExZTItODBiMC0wMDUwNTZiYzQ3
OWYxJjAkBgkqhkiG9w0BCQEWF2pmZWRAd2FsbDMudGVzdC5pYmJ0LmJlMIGfMA0G
CSqGSIb3DQEBAQUAA4GNADCBiQKBgQDEqQrtZgcWutGDlMs6RkkA8+tVJs1HU1Jx
nj6ZdRWf7Y9oPa47T5uUgaEm6G2N7dVuaNjOlEA8kVWWZvME8msBEyTGxUD15ZxF
Lyah15R4MCaQgYXz+yT3G2wIWFz5qB75XQXPsxRN9IlXIPwFT7Wjz9JEbQkU7Nlr
VTmcTdgapQIDAQABo4HlMIHiMAwGA1UdEwEB/wQCMAAwHQYDVR0OBBYEFBuCLMrE
Q47dZxwoRFAhijCmL5OWMFEGA1UdEQRKMEiGLXVybjpwdWJsaWNpZDpJRE4rd2Fs
bDMudGVzdC5pYmJ0LmJlK3VzZXIramZlZIEXamZlZEB3YWxsMy50ZXN0LmliYnQu
YmUwYAYIKwYBBQUHAQEEVDBSMFAGFGmDzJOAqJjMqMe9saeAgKqu14obhjhodHRw
czovL3d3dy53YWxsMy50ZXN0LmliYnQuYmU6MTIzNjkvcHJvdG9nZW5pL3htbHJw
```

```
Yy9zYTANBgkqhkiG9w0BAQQFAAOBgQBMwDIGIYN6ZcS17ZnvINsx6MLmso0LSTkN
1jKGvPDsM2tlCULsy6CFGzpG4i2OGzUAFLph+T2O3MkA50YJTcjs7QId7sHTIpum
77dpGxdn508Ua97ip9rcYlZ/JvGe/LdvghwfAidG8SKIyYRRhM33iv9n5GEHNpNy
dvnK0DRJ/g==
</owner_gid>
 <owner_urn>urn:publicid:IDN+wall3.test.ibbt.be+user+jfed</owner_urn>
 <target_gid>MIIDbTCCAtagAwIBAgICA1YwDQYJKoZIhvcNAQEEBQAwgbkxCzAJBgNVBAYTAkJF
MQwwCgYDVQQIEwNPVkwxDjAMBgNVBAcTBUdoZW50MR4wHAYDVQQKExVVR2VudC1J
bnRlYy1JQkNOL0lCQlQxHjAcBgNVBAsTFUNlcnRpZmljYXRlIEF1dGhvcml0eTEg
MB4GA1UEAxMXYm9zcy53YWxsMy50ZXN0LmliYnQuYmUxKjAoBgkqhkiG9w0BCQEW
G3Z3YWxsLW9wc0BhdGxhbnRpcy51Z2VudC5iZTAeFw0xMzA1MDIxNzIxMTJaFw0x
ODEwMjMxODIxMTJaMIG6MQswCQYDVQQGEwJCRTEMMAoGA1UECBMDT1ZMMR4wHAYD
VQQKExVVR2VudC1JbnRlYy1JQkNOL0lCQlQxIzAhBgNVBAsTGndhbGwzZ2VuaS5k
ZW1vc3ViYXV0aG9yaXR5MS0wKwYDVQQDEyQxMTdhZGUyNi1iMzU1LTExZTItODBi
MC0wMDUwNTZiYzQ3OWYxKTAnBgkqhkiG9w0BCQEWGmJyZWNodC52ZXJtZXVsZW5A
aW1pbmRzLmJlMIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCoMaf/jZRRGFbI
us5cPrHS2I1q8K4hGcCEWFPQESg1KS8pe8II4Zzv+Q0bB8+28ECF/avrVOgUIKVd
H6LH9qHIBS02v7XcOlsOhyIp0C6AMt9YJe6DUOAyHL3ePaEZ7WGhlzyL8kfXieR1
7eQLSz3QAwdfrj5Re0Y2FFAa0htbvwIDAQABo4GAMH4wHQYDVR0OBBYEFIC+vg+M
AUayuXkTZSVk9hKJKF/rME8GA1UdEQRIMEaGRHVybjpwdWJsaWNpZDpJRE4rd2Fs
bDMudGVzdC5pYmJ0LmJlOmpmZWR1c2VycytzbGljZStkZW1vc3ViYXV0aG9yaXR5
MAwGA1UdEwEB/wQCMAAwDQYJKoZIhvcNAQEEBQADgYEAg1IOS13GViRRgMcqIYwz
b1ALKds7OzeXgba3MnjJ+67ZUadxFyvvcv0o+CESGq9FFpbXS5X688nuTYUxPTRc
HS27liaLy0IIFnh908Q87NMorIyXFkO8/232P1P+0tynGtVCexQcQRUFKauJ1S7O
/P9CouSgNfv6XSc/UyJrL/w=
</target_gid>
 <target_urn>urn:publicid:IDN+wall3.test.ibbt.be:jfedusers+slice+demosubauthority</target_urn>
 <uuid>118d7e15-b355-11e2-80b0-005056bc479f</uuid>
 <expires>2013-05-03T00:21:12Z</expires>
```


Interpret slice credential

If you put a
-----BEGIN CERTIFICATE-----
-----END CERTIFICATE-----
around the gid, you can look into them with
openssl x509 -text -in gid1.pem

User credential

```
<owner_urn>urn:publicid:IDN+wall3.test.ibbt.be+user+jfed</owner_urn>
```

Below you see that the slice authority has signed the user certificate, and you can find an email address of the slice authority administrators and the user (jfed@wall3.test.ibbt.be)
Certificate:
Data:
Version: 3 (0x2)
Serial Number: 831 (0x33f)
Signature Algorithm: md5WithRSAEncryption
Issuer: C=BE, ST=OVL, L=Ghent, O=UGent-Intec-IBCN/IBBT, OU=Certificate Authority, CN=boss.wall3.test.ibbt.be/emailAddress=vwall-ops@atlantis.ugent.be
Validity
Not Before: May 1 14:28:04 2013 GMT
Not After : May 1 14:28:04 2014 GMT
Subject: C=BE, ST=OVL, O=UGent-Intec-IBCN/IBBT, OU=wall3geni.jfed, CN=b79e21a1-b26a-11e2-80b0-005056bc479f/emailAddress=jfed@wall3.test.ibbt.be
Subject Public Key Info:
Public Key Algorithm: rsaEncryption
RSA Public Key: (1024 bit)

```
            Modulus (1024 bit):
                00:c4:a9:0a:ed:66:07:16:ba:d1:83:94:cb:3a:46:
                49:00:f3:eb:55:26:cd:47:53:52:71:9e:3e:99:75:
                15:9f:ed:8f:68:3d:ae:3b:4f:9b:94:81:a1:26:e8:
                6d:8d:ed:d5:6e:68:d8:ce:94:40:3c:91:55:96:66:
                f3:04:f2:6b:01:13:24:c6:c5:40:f5:e5:9c:45:2f:
                26:a1:d7:94:78:30:26:90:81:85:f3:fb:24:f7:1b:
                6c:08:58:5c:f9:a8:1e:f9:5d:05:cf:b3:14:4d:f4:
                89:57:20:fc:05:4f:b5:a3:cf:d2:44:6d:09:14:ec:
                d9:6b:55:39:9c:4d:d8:1a:a5
            Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Basic Constraints: critical
                CA:FALSE
            X509v3 Subject Key Identifier:
                1B:82:2C:CA:C4:43:8E:DD:67:1C:28:44:50:21:8A:30:A6:2F:93:96
            X509v3 Subject Alternative Name:
                URI:urn:publicid:IDN+wall3.test.ibbt.be+user+jfed, email:jfed@wall3.test.ibbt.be
            Authority Information Access:
                2.25.305821105408246119474742976030998643995                -
URI:https://www.wall3.test.ibbt.be:12369/protogeni/xmlrpc/sa

    Signature Algorithm: md5WithRSAEncryption
        4c:c0:32:06:21:83:7a:65:c4:b5:ed:99:ef:20:db:31:e8:c2:
        e6:b2:8d:0b:49:39:0d:d6:32:86:bc:f0:ec:33:6b:65:09:42:
        ec:cb:a0:85:1b:3a:46:e2:2d:8e:1b:35:00:14:ba:61:f9:3d:
        8e:dc:c9:00:e7:46:09:4d:c8:ec:ed:02:1d:ee:c1:d3:22:9b:
        a6:ef:b7:69:1b:17:67:e7:4f:14:6b:de:e2:a7:da:dc:62:56:
        7f:26:f1:9e:fc:b7:6f:82:1c:1f:02:27:46:f1:22:88:c9:84:
        51:84:cd:f7:8a:ff:67:e4:61:07:36:93:72:76:f9:ca:d0:34:
        49:fe
-----BEGIN CERTIFICATE-----
MIIDwzCCAyygAwIBAgICAz8wDQYJKoZIhvcNAQEEBQAwgbkxCzAJBgNVBAYTAkJF
MQwwCgYDVQQIEwNPVkwxDjAMBgNVBAcTBUdoZW50MR4wHAYDVQQKExVVR2VudC1J
bnRlYy1JQkNOL0lCQlQxHjAcBgNVBAsTFUNlcnRpZmljYXRlIEF1dGhvcml0eTEg
MB4GA1UEAxMXYm9zcy53YWxsMy50ZXN0LmliYnQuYmUxKjAoBgkqhkiG9w0BCQEW
G3Z3YWxsLW9wc0BhdGxhbnRpcy51Z2VudC5iZTAeFw0xMzA1MDExNDI4MDRaFw0x
NDA1MDExNDI4MDRaMIGrMQswCQYDVQQGEwJCRTEMMAoGA1UECBMDT1ZMMR4wHAYD
VQQKExVVR2VudC1JbnRlYy1JQkNOL0lCQlQxFzAVBgNVBAsTDndhbGwzZ2VuaS5q
ZmVkMS0wKwYDVQQDEyRiNzllMjFhMS1iMjZhLTExZTItODBiMC0wMDUwNTZiYzQ3
OWYxJjAkBgkqhkiG9w0BCQEWF2pmZWRAd2FsbDMudGVzdC5pYmJ0LmJlMIGfMA0G
CSqGSIb3DQEBAQUAA4GNADCBiQKBgQDEqQrtZgcWutGDlMs6RkkA8+tVJs1HU1Jx
nj6ZdRWf7Y9oPa47T5uUgaEm6G2N7dVuaNjOlEA8kVWWZvME8msBEyTGxUD15ZxF
Lyah15R4MCaQgYXz+yT3G2wIWFz5qB75XQXPsxRN9IlXIPwFT7Wjz9JEbQkU7Nlr
VTmcTdgapQIDAQABo4HlMIHiMAwGA1UdEwEB/wQCMAAwHQYDVR0OBBYEFBuCLMrE
Q47dZxwoRFAhijCmL5OWMFEGA1UdEQRKMEiGLXVybjpwdWJsaWNpZDpJRE4rd2Fs
bDMudGVzdC5pYmJ0LmJlK3VzZXIramZlZIEXamZlZEB3YWxsMy50ZXN0LmliYnQu
YmUwYAYIKwYBBQUHAQEEVDBSMFAGFGmDzJOAqJjMqMe9saeAgKqu14obhjhodHRw
czovL3d3dy53YWxsMy50ZXN0LmliYnQuYmU6MTIzNjkvcHJvdG9nZW5pL3htbHJw
Yy9zYTANBgkqhkiG9w0BAQQFAAOBgQBMwDIGIYN6ZcS17ZnvINsx6MLmso0LSTkN
```

1jKGvPDsM2tlCULsy6CFGzpG4i2OGzUAFLph+T2O3MkA50YJTcjs7QId7sHTIpum
77dpGxdn508Ua97ip9rcYlZ/JvGe/LdvghwfAidG8SKIyYRRhM33iv9n5GEHNpNy
dvnK0DRJ/g==
-----END CERTIFICATE-----

Slice credential

```
<target_urn>urn:publicid:IDN+wall3.test.ibbt.be:jfedusers+slice+demosubauthority</target_urn>
<expires>2013-05-03T00:21:12Z</expires>
```

In the URN (:jfedusers), you see that it is a slice 'demosubauthority' within the subauthority jfedusers of wall3.test.ibbt.be.

Certificate:
  Data:
    Version: 3 (0x2)
    Serial Number: 854 (0x356)
    Signature Algorithm: md5WithRSAEncryption
    Issuer: C=BE, ST=OVL, L=Ghent, O=UGent-Intec-IBCN/IBBT, OU=Certificate Authority, CN=boss.wall3.test.ibbt.be/emailAddress=vwall-ops@atlantis.ugent.be
    Validity
      Not Before: May 2 17:21:12 2013 GMT
      Not After : Oct 23 18:21:12 2018 GMT
    Subject: C=BE, ST=OVL, O=UGent-Intec-IBCN/IBBT, OU=wall3geni.demosubauthority, CN=117ade26-b355-11e2-80b0-005056bc479f/emailAddress=brecht.vermeulen@iminds.be
    Subject Public Key Info:
      Public Key Algorithm: rsaEncryption
      RSA Public Key: (1024 bit)
        Modulus (1024 bit):
          00:a8:31:a7:ff:8d:94:51:18:56:c8:ba:ce:5c:3e:
          b1:d2:d8:8d:6a:f0:ae:21:19:c0:84:58:53:d0:11:
          28:35:29:2f:29:7b:c2:08:e1:9c:ef:f9:0d:1b:07:
          cf:b6:f0:40:85:fd:ab:eb:54:e8:14:20:a5:5d:1f:
          a2:c7:f6:a1:c8:05:2d:36:bf:b5:dc:3a:5b:0e:87:
          22:29:d0:2e:80:32:df:58:25:ee:83:50:e0:32:1c:
          bd:de:3d:a1:19:ed:61:a1:97:3c:8b:f2:47:d7:89:
          e4:75:ed:e4:0b:4b:3d:d0:03:07:5f:ae:3e:51:7b:
          46:36:14:50:1a:d2:1b:5b:bf
        Exponent: 65537 (0x10001)
    X509v3 extensions:
      X509v3 Subject Key Identifier:
        80:BE:BE:0F:8C:01:46:B2:B9:79:13:65:25:64:F6:12:89:28:5F:EB
      X509v3 Subject Alternative Name:
        URI:urn:publicid:IDN+wall3.test.ibbt.be:jfedusers+slice+demosubauthority
      X509v3 Basic Constraints: critical
        CA:FALSE
  Signature Algorithm: md5WithRSAEncryption
    83:52:0e:4b:5d:c6:56:24:51:80:c7:2a:21:8c:33:6f:50:0b:
    29:db:3b:3b:37:97:81:b6:b7:32:78:c9:fb:ae:d9:51:a7:71:
    17:2b:ef:72:fd:28:f8:21:12:1a:af:45:16:96:d7:4b:95:fa:
    f3:c9:ee:4d:85:31:3d:34:5c:1d:2d:bb:96:26:8b:cb:42:08:

16:78:7d:d3:c4:3c:ec:d3:28:ac:8c:97:16:43:bc:ff:6d:f6:
3f:53:fe:d2:dc:a7:1a:d5:42:7b:14:1c:41:15:05:29:ab:89:
d5:2e:ce:fc:ff:42:a2:e4:a0:35:fb:fa:5d:27:3f:53:22:6b:
2f:fc
-----BEGIN CERTIFICATE-----
MIIDbTCCAtagAwIBAgICA1YwDQYJKoZIhvcNAQEEBQAwgbkxCzAJBgNVBAYTAkJF
MQwwCgYDVQQIEwNPVkwxDjAMBgNVBAcTBUdoZW50MR4wHAYDVQQKExVVR2VudC1J
bnRlYy1JQkNOL0lCQlQxHjAcBgNVBAsTFUNlcnRpZmljYXRlIEF1dGhvcml0eTEg
MB4GA1UEAxMXYm9zcy53YWxsMy50ZXN0LmliYnQuYmUxKjAoBgkqhkiG9w0BCQEW
G3Z3YWxsLW9wc0BhdGxhbnRpcy51Z2VudC5iZTAeFw0xMzA1MDIxNzIxMTJaFw0x
ODEwMjMxODIxMTJaMIG6MQswCQYDVQQGEwJCRTEMMAoGA1UECBMDT1ZMMR4wHAYD
VQQKExVVR2VudC1JbnRlYy1JQkNOL0lCQlQxIzAhBgNVBAsTGndhbGwzZ2VuaS5k
ZW1vc3ViYXV0aG9yaXR5MS0wKwYDVQQDEyQxMTdhZGUyNi1iMzU1LTExZTItODBi
MC0wMDUwNTZiYzQ3OWYxKTAnBgkqhkiG9w0BCQEWGmJyZWNodC52ZXJtZXVsZW5A
aW1pbmRzLmJlMIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCoMaf/jZRRGFbI
us5cPrHS2I1q8K4hGcCEWFPQESg1KS8pe8II4Zzv+Q0bB8+28ECF/avrVOgUIKVd
H6LH9qHIBS02v7XcOlsOhyIp0C6AMt9YJe6DUOAyHL3ePaEZ7WGhlzyL8kfXieR1
7eQLSz3QAwdfrj5Re0Y2FFAa0htbvwIDAQABo4GAMH4wHQYDVR0OBBYEFIC+vg+M
AUayuXkTZSVk9hKJKF/rME8GA1UdEQRIMEaGRHVybjpwdWJsaWNpZDpJRE4rd2Fs
bDMudGVzdC5pYmJ0LmJlOmpmZWR1c2VycytzbGljZStkZW1vc3ViYXV0aG9yaXR5
MAwGA1UdEwEB/wQCMAAwDQYJKoZIhvcNAQEEBQADgYEAg1IOS13GViRRgMcqIYwz
b1ALKds7OzeXgba3MnjJ+67ZUadxFyvvcv0o+CESGq9FFpbXS5X688nuTYUxPTRc
HS27liaLy0IIFnh908Q87NMorIyXFkO8/232P1P+0tynGtVCexQcQRUFKauJ1S7O
/P9CouSgNfv6XSc/UyJrL/w=
-----END CERTIFICATE-----

Appendix G: Evaluation of possible architectural approaches for SLA management

Two different approaches for the implementation of SLA management in Fed4FIRE have been considered. Both are similar as far as requirement coverage is concerned but they have different implications for the federation and the testbeds, which will be analysed the remainder of this subsection.

Before we go deeper into the different architectural possibilities, we should also clarify that, regardless of the architecture chosen, the SLA mangement itself can be centralised or distributed, depending on how thick the federation layer is. In the case of Fed4FIRE, with a multi-provider, multi-technology and multi-resource/service situation within only one experiment, both are possible:

- **Fed4FIRE provides federated centralized SLAs**: An SLA is agreed between the experimenter and the federation as an entity. The federation deals with underlying testbeds as a sort of SLA broker.
- **Fed4FIRE provides federated SLA setup**: An SLA is agreed between the experimenter and every single testbed involved in an experiment having adopted SLAs. The federation ensures there is an SLA established with every testbed before the experiment begins.
  In this case, on an experiment basis, the SLAs should be agreed between the experimenter and all the SLA-enabled testbeds involved in the experiment. This means that the SLAs are not agreed between the experimenter and some federation entity. Instead, the federated SLA management makes sure that, before an experiment begins, all required SLAs have been agreed between the involved parties. The main advantages of this approach, as far as we see it, are:
  - The federation layer is kept as thin as possible like this, which is in line with the current WP2 approach.
  - There are no billing/conciliation functionalities in Fed4FIRE federation for the moment (some testbeds intend to operate commercially; those testbeds will deal directly with experimenters for billing and conciliation).
- **Fed4FIRE provides a federated SLA tool**: This approach is related to the previous one (Fed4FIRE provides federated SLA setup), but in this case the federation does not ensure that there is an SLA established with every testbed before the experiment begins. Every testbed having adopted SLAs will expose its SLAs and will manage its SLA commitments individually. With this approach, the federation only provides the set of tools for a testbed to implement SLA management. It's up to the testbed to use it or not.

The following table gathers a comparison of these alternatives.

	Advantages	Disadvantages	Selected as final approach
Federated centralized SLA	• Easier for the experimenter	• More complicated to implement	
Federated SLA setup	• Fits Fed4FIRE philosophy (light federation layer)	• Less easy (more bureaucratic) for the experimenter	
Federated SLA tool	• Fits Fed4FIRE philosophy (light federation layer)	• More bureaucratic for the experimenter and	X

	• Testbeds have the freedom to implement it or not. • Less modifications of the existing tool are required since currently it only works for one provider	more difficult to implement for the testbeds	

Table 2: Comparison of approaches for implementing SLA Management

Taking into account a federated SLA tool as the most suitable approach for Fed4FIRE, the different architectural possibilities can be presented.

Centralised approach

In the centralised approach, the SLA management is a centralised component of the architecture. This can be implemented by considering the SLA management module like an upper layer to SFA. In this case, there is only one SLA Management module, which has advantages and disadvantages that are detailed bellow:

- Advantages:
  - There is only one SLA Management module in Fed4FIRE, which is easier for maintenance.
  - This option requires less implementation effort from testbeds.
  - The experimenter retrieves SLAs from only one centralised place.

- Disadvantages:
  - It implies important modifications in the state-of-the-art SLA solution (which does not currently operate in a multi-provider environment) and integration within a testbed whenever a new testbed is incorporated to the federation, depending on the monitoring system used by this testbed and whether it is SFA compliant or not.
  - The needed communication with the Aggregator Manager (AM) of each testbed and the possible different monitoring agents is complex.

Regarding the current technical situation, this approach implies:

- The project's MySlice portal and other experimenter tools (e.g. jFed, Omni, SFI, Flack) need to be able to display SLA options for an experimenter. In the particular case of the portal, new MySlice plugins to be integrated in existing forms are required.
- The SLA Management module needs to develop new SFA calls in order to be able to provide the following functionalities: publication, discovery and negotiation of SLAs requirements. That is to say, through this interface, SLA management communicates with the AM of the different testbeds and gets information about which SLAs and resources are offered by each testbed, and even just before doing the SLA negotiation between both parties (experimenter and testbed), it may allow knowing if the resources required by the experimenter are available or not.
- API Rest: this API is needed for the creation and the management of SLAs.
- OML Interface: to communicate with the different monitoring agents in order to obtain monitoring information.

Distributed approach

The second option considered is to include the SLA management as a distributed component inside each testbed. The distributed approach has advantages and disadvantages that are detailed bellow.

- Advantages:
  - Thin federation layer.
  - Flexibility: it's up to every testbed to incorporate the SLA module.
  - This option requires less modifications in the existing SLA solution since it currently deals with only one provider.
  - It is easier to adapt within the PDP mechanisms deployed at each testbed.
  - It allows a testbed to use the SLA management outside of Fed4FIRE for its own purposes.

- Disadvantages:
  - It requires more implementation effort from the testbeds.
  - This option requires more coordination between testbeds in order for all implementations to operate in a federated and homogeneous manner. Otherwise the interaction for the experimenter would be more difficult.

Regarding the current technical situation, this approach implies:

- The project's MySlice portal and other experimenter tools need to be able to display SLA options for an experimenter. In the particular case of the portal, new MySlice plugins needs to be implemented.
- A new catalogue database containing all the available resources/services offered by the different testbeds is required in every testbed. This catalogue needs to be interfaced with client tools.
- Each testbed adopting SLAs manages them in a distributed manner. There is no aggregated processing of SLAs at a federated level.
- The experimenter agrees only one general SLA per experiment, which is split in as many SLAs as testbeds are involved in the experiment. In the end, there will be one SLA between the experimenter and each testbed involved in the experiment.
- The experimenter gets the SLA evaluation information individually from every testbed, for all testbeds with SLA mechanisms involved in the experiment. Aggregation of results could be tackled in cycle 3.

Comparison

The following table summarizes the main advantages and disadvantages of these different solutions:

Approach	Advantages	Disadvantages	Selected as final approach
Centralized approach	• Easier for maintenance (only one SLA management module) • Experimenter retrieves SLAs from only one centralised location.	• More complicated to implement (communication with all the AM of each testbed and all the possible different monitoring agents). • It does not fit Fed4FIRE 's WP2 philosophy (more complex federation layer)	

Approach	Advantages	Disadvantages	Selected as final approach
Distributed approach	• Thin federation layer • Flexibility: it's up to every testbed to incorporate the SLA module. • Less modifications in the existing SLA tool (it currently deals with only one provider) • It is easier to adapt within the PDP mechanisms deployed at each testbed.	• More implementation effort from testbeds in order to operate in a federated and homogeneous manner.	X

Table 3: Comparison of architectural approaches for implementing SLA Management

The distributed approach seems to be more suitable and flexible for cycle 2 of Fed4FIRE. Besides, this solution can evolve towards a more centralized management approach (by aggregation) in case it is required in the future.

Appendix H: Workflow for registering an account and getting a certificate

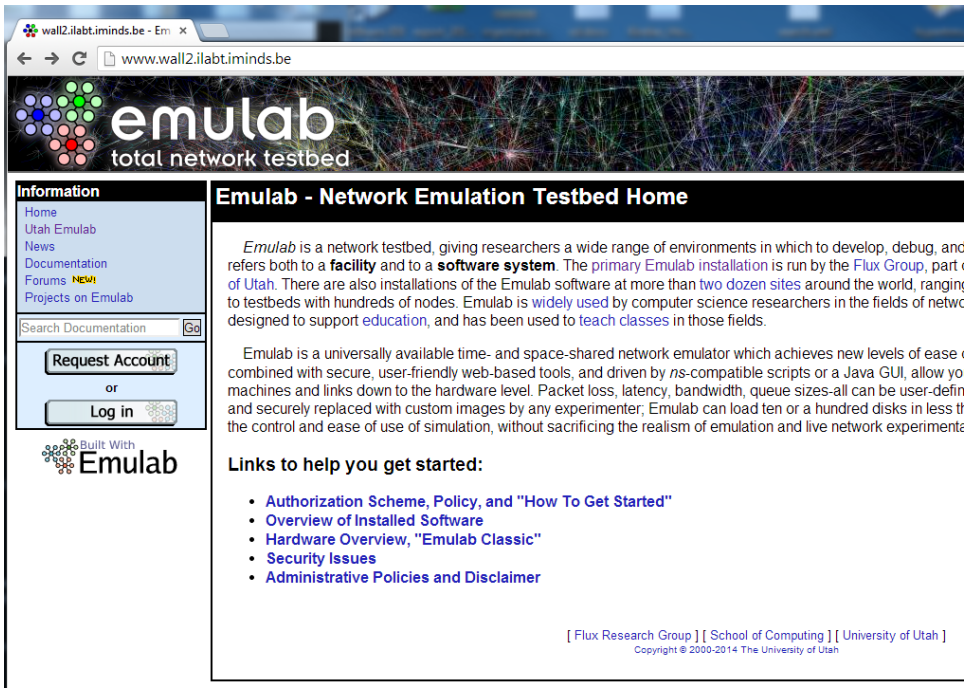
This appendix shows one of the possible workflows to get an account and certificate from a Fed4FIRE authority. Note that this presented workflow is intended as an illustration of the architectural concepts described in this deliverable, to make them as clear as possible. The actual content of this appendix can be situated somewhere between architectural descriptions and more detailed specifications. Therefore it is likely that this information (possibly in more detail) will also become part of WP5's specifications for cycle 2 in deliverable D5.2.

Introduction

Fed4FIRE works with X.509 certificates to authenticate and authorize experimenters on testbeds. In this section we will guide you through creating such a certificate. Testbeds can decide which certificates from which authorities they accept and as such federations of testbeds and authorities are established. Fed4FIRE is such a federation.
In the end, you will have a file on your local PC which contains this certificate information and this can then be used in experimenter tools.
This procedure has to be run only one.

Register for an account

The current authority which can be used in the Fed4FIRE federation is located at http://www.wall2.ilabt.iminds.be. (in the future, this will also be possible through the Fed4FIRE portal, we will then update this documentation).



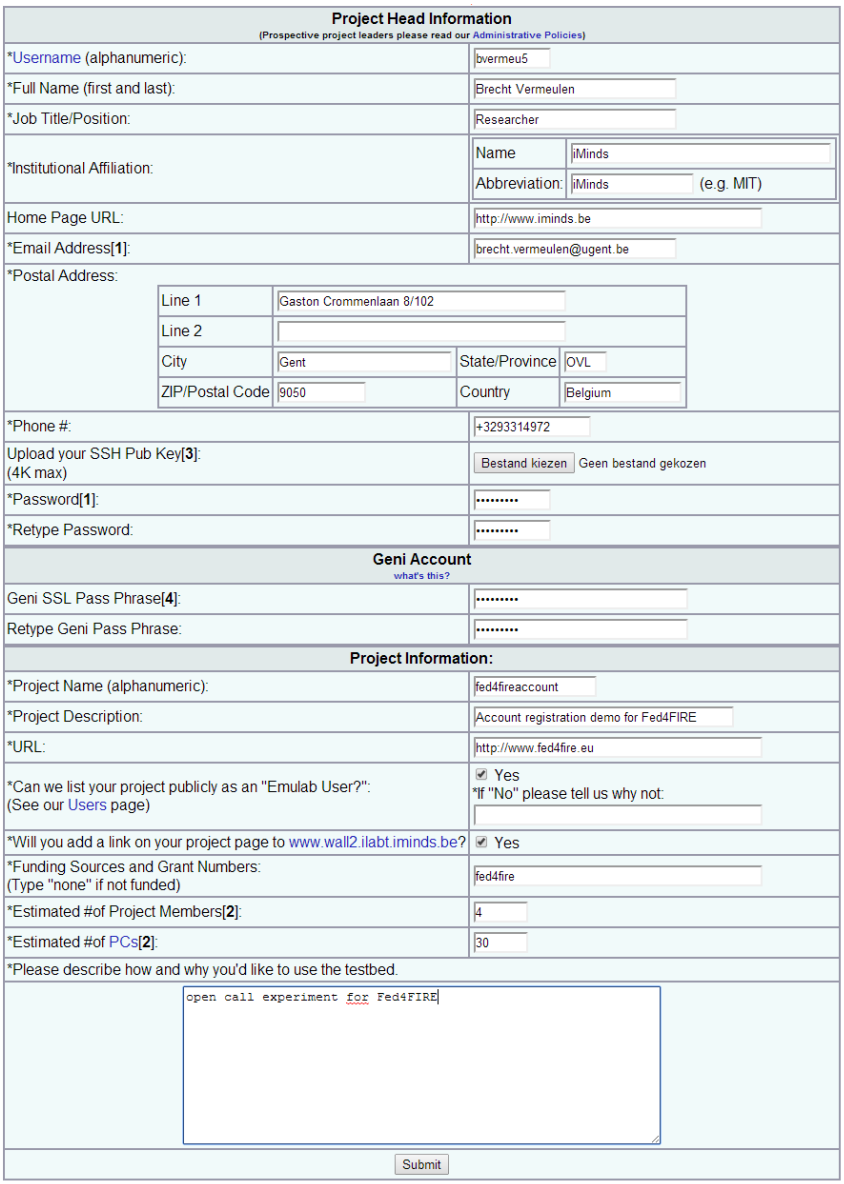
At the left side, we click on `Request Account`.

The authority has the concept of Projects which bundle multiple experimenters. In a Project, the PIs (Principal Investigators) can approve new experimenters for that project, without Fed4FIRE administrators needing to approve this.

Here we will start with creating a new project by clicking Start a New Project. If you have been invited to a Project, just click Join an Existing Project (or you might have received a specific URL). Now, we have to fill in our account information. There are numerous checks (e.g. passphrase should be longer then 10 characters, home page URLs should be reachable and so on), so fill in as correct as possible.

Please be aware of the following:

- enter a phone number you can remember. This is used if you want to recover the password for this authority portal.
- the password field is the password that will be used for logging in on this authority portal.
- the passphrase (in the section GENI Account) is the passphrase for your certificate and should be filled.



Project Head Information
(Prospective project leaders please read our Administrative Policies)

*Username (alphanumeric):	bvermeu5
*Full Name (first and last):	Brecht Vermeulen
*Job Title/Position:	Researcher
*Institutional Affiliation:	Name: iMinds Abbreviation: iMinds (e.g. MIT)
Home Page URL:	http://www.iminds.be
*Email Address[1]:	brecht.vermeulen@ugent.be
*Postal Address:	Line 1: Gaston Crommenlaan 8/102 Line 2: City: Gent — State/Province: OVL ZIP/Postal Code: 9050 — Country: Belgium
*Phone #:	+3293314972
Upload your SSH Pub Key[3]: (4K max)	Bestand kiezen — Geen bestand gekozen
*Password[1]:	•••••••••
*Retype Password:	•••••••••

Geni Account
what's this?

Geni SSL Pass Phrase[4]:	•••••••••
Retype Geni Pass Phrase:	•••••••••

Project Information:

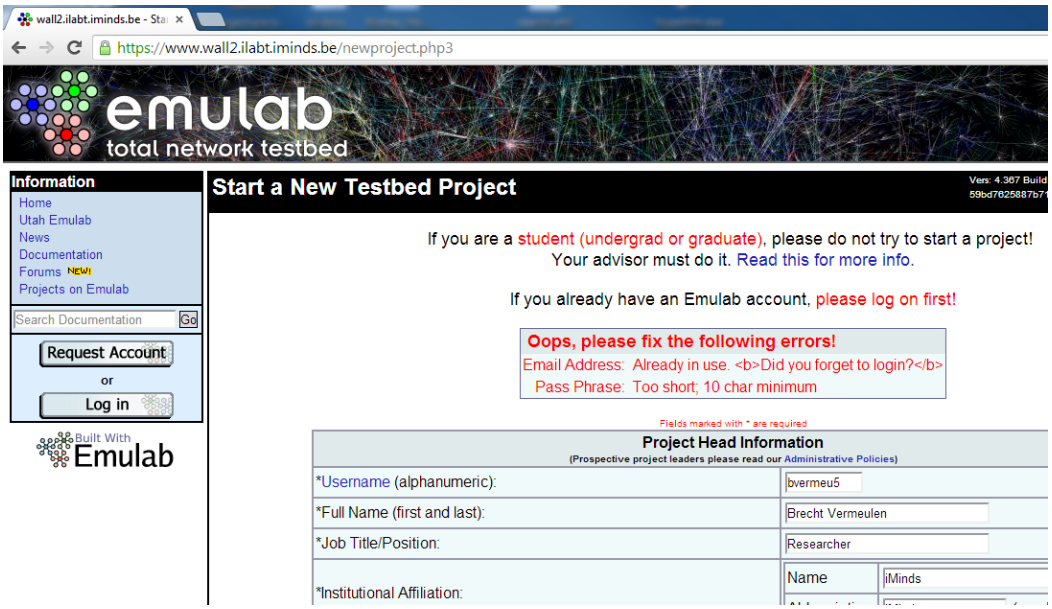
*Project Name (alphanumeric):	fed4fireaccount
*Project Description:	Account registration demo for Fed4FIRE
*URL:	http://www.fed4fire.eu
*Can we list your project publicly as an "Emulab User?": (See our Users page)	Yes *If "No" please tell us why not:
*Will you add a link on your project page to www.wall2.ilabt.iminds.be?	Yes
*Funding Sources and Grant Numbers: (Type "none" if not funded)	fed4fire
*Estimated #of Project Members[2]:	4
*Estimated #of PCs[2]:	30

*Please describe how and why you'd like to use the testbed.

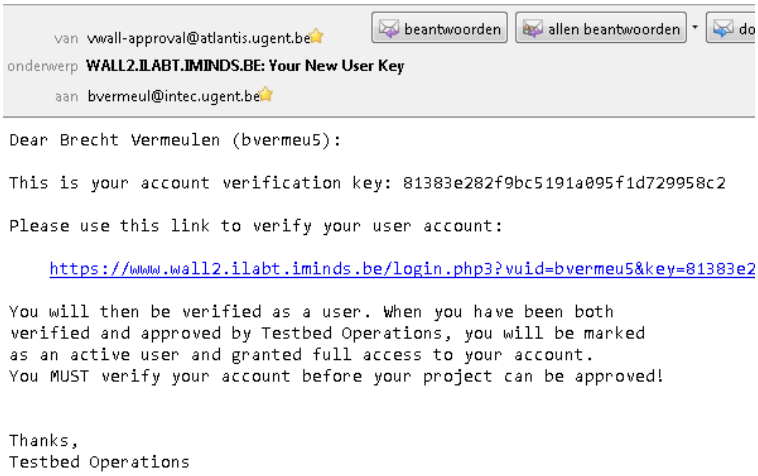
open call experiment for Fed4FIRE

Submit

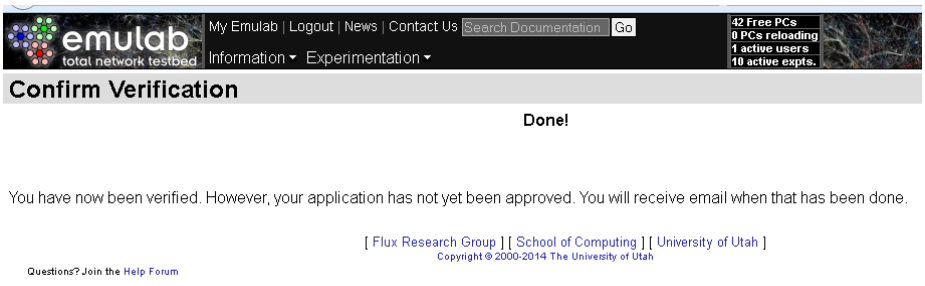
If there are some problems when submitting the form, please scroll up to the top, where the errors are mentioned, see example below.



If all goes well, you will receive an email in your mailbox to verify your account. Click on the link in the email. You might have to login with your authority portal account you just created.



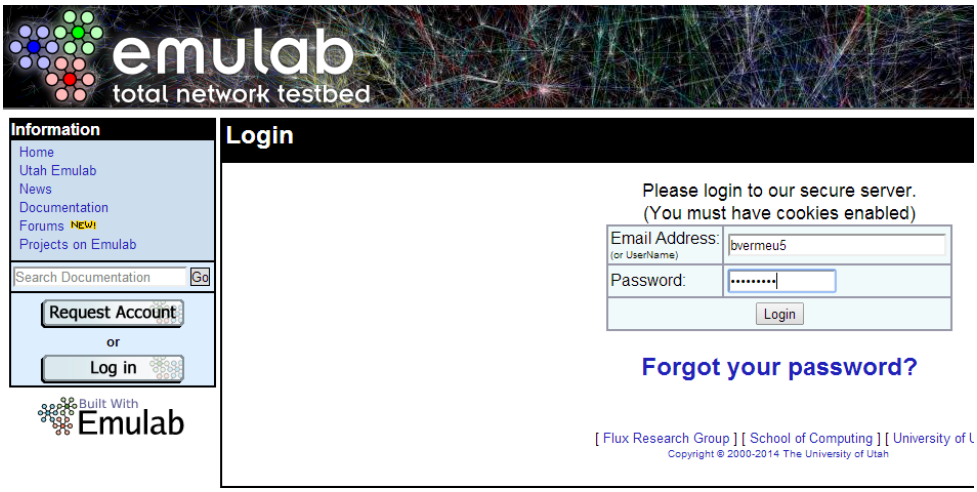
If the verification on the authority portal goes well, you'll see this:



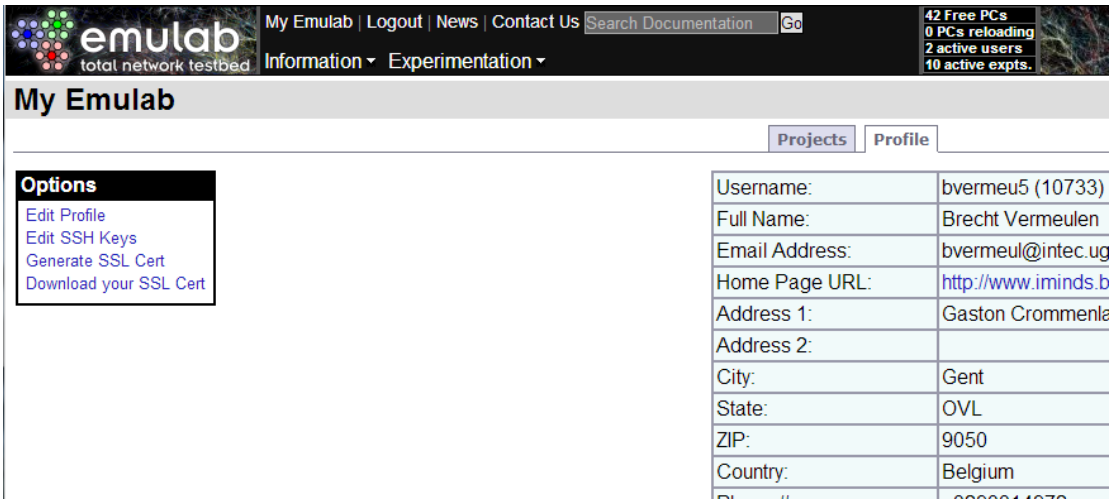
Now, it can take some time to have a Fed4FIRE administrator or project administrator approved your request. You will receive an email when that is done. After that, you can login on the authority portal.

Download your Fed4FIRE certificate

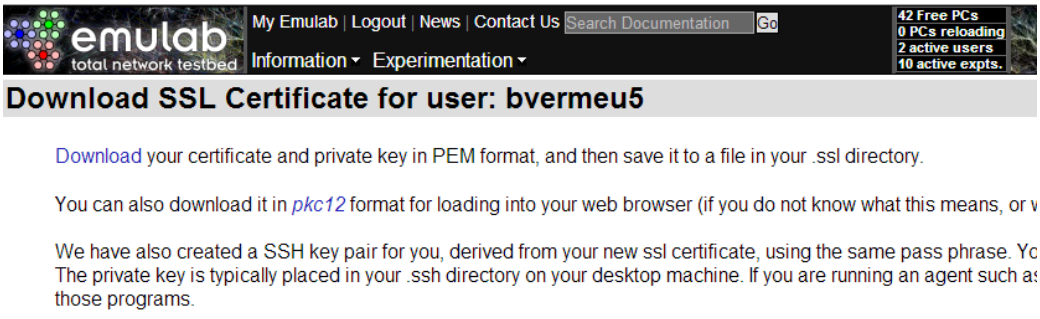
Login on the authority portal.



Now, we will download the approved certificate. At the left side, click `Download your SSL Cert`.



You'll land on this page, and then click the first Download for downloading your Fed4FIRE Certificate in PEM format.



You'll see the text from the certificate (as screenshot below), and just save this to a local file on your PC, e.g.`fed4fire_youraccount.pem`.

```
-----BEGIN RSA PRIVATE KEY-
Proc-Type: 4,ENCRYPTED
DEK-Info: DES-EDE3-CBC,0F93

KLBpASSO3FiWhwkWR4uSnDoT9ow
c2plsD3JtZEx+jRJHxSebepGKD/
MgeoNadXbhcBO3+ub6NHQLUZnBS
Qu6mCBqGe8ossCc7HVZPRdglok7
S1gLzminYbkxEWOhZrZRl+0jUuY
yXMQNeuvAuLVpRirlcg/1vWAopX
L29vfhJNjj4DO5WiCUs7jiGHsWH
PRH/5xI/kIeHOUZJEKG6o9muwLm
0JbtK4iq1I+Y5BAem7+K/8TpbKT
NEYGJK4pA69Sa2EhJo34TcI+Oxs
XbLIDzKG5Q0pkRwanji+RcQLokX
6yozngUEN4T7hqiWtt+H6mtpDOd
IH5z8rGJmkKaRw2hADQSJ9x7K5g
-----END RSA PRIVATE KEY---
-----BEGIN CERTIFICATE-----
MIIECzCCA3SgAwIBAgICJrEwDQY
MQswCQYDVQQIEwJPVjEOMAwGA1U
IGlsYWIudDEeMBwGA1UECxMVQ2V
Expib3NzLndhbGwyLmlsYWJ0Lml
bGwtb3BzQGF0bGFudGlzLnVnZW5
NDA0NDIzOFowgbIxCzAJBgNVBAY
aU1pbmRzIC0gaWxhYi50MR4wHAY
```

Appendix I: Workflow for creating an SSH key on different operating systems

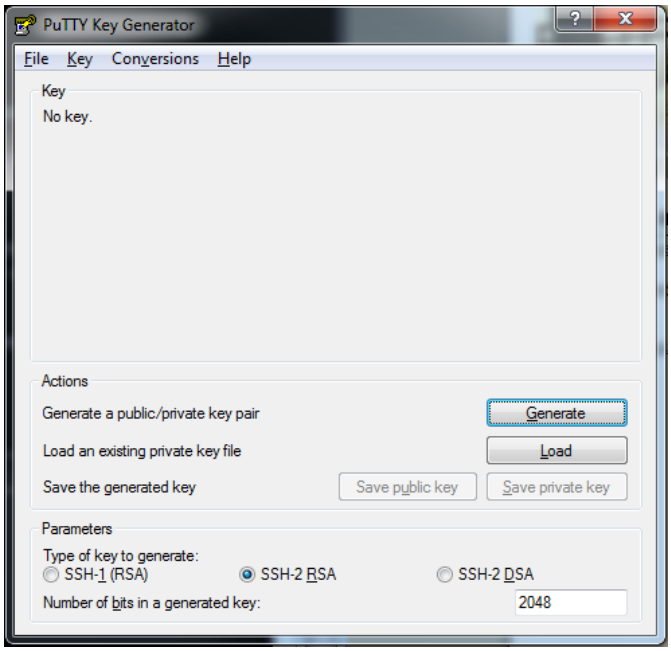
Introduction

In Fed4FIRE, access to testbed resources is done through SSH and more specifically SSH based on public/private key authentication. This section will guide you through the process of generating an SSH key. If you already have an SSH key, you can skip this. Note that this presented workflow is intended as an illustration of the architectural concepts described in this deliverable, to make them as clear as possible. The actual content of this appendix can be situated somewhere between architectural descriptions and more detailed specifications. Therefore it is likely that this information (possibly in more detail) will also become part of WP5's specifications for cycle 2 in deliverable D5.2.

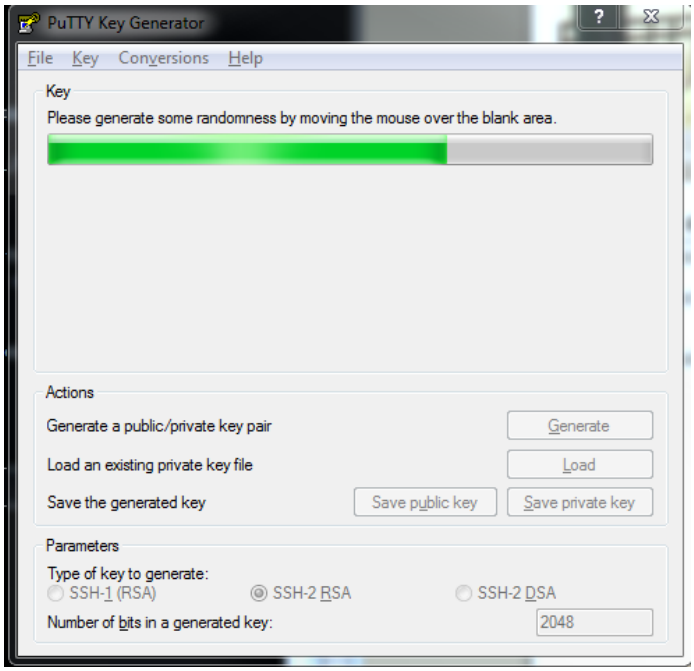
Windows platform: PuTTY

PuTTY is a widely used SSH client for Windows and it includes the tool PuTTYgen to create an SSH key. You can download PuTTY from http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html and you can download an installer of the latest version here.

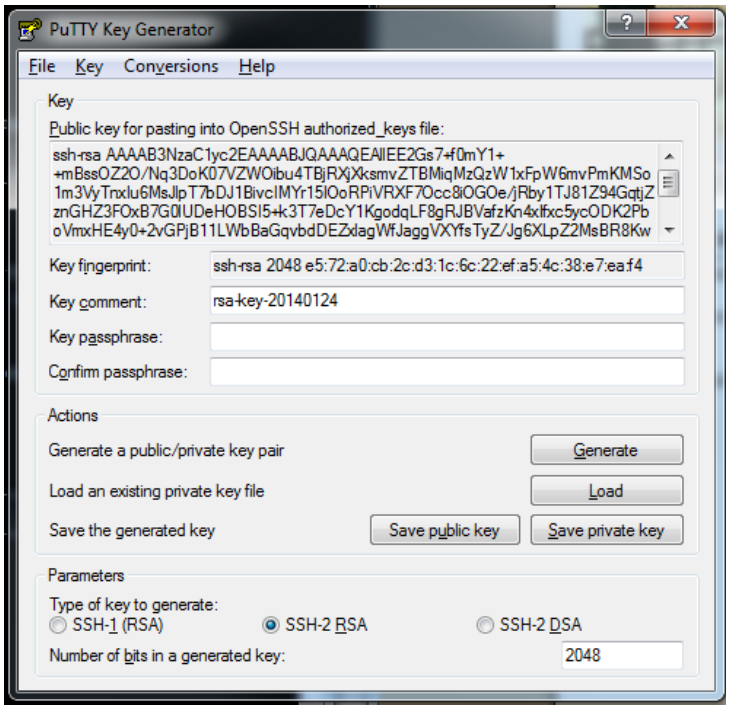
To generate an SSH key, start the tool PuTTYgen.



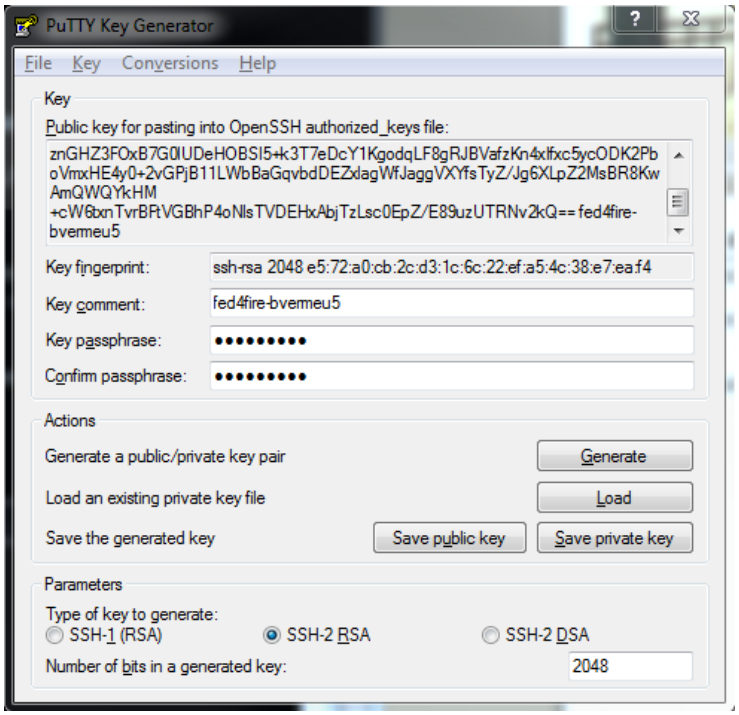
If you click on `Generate`, it will start collecting random information while you move your mouse over the blank session as indicated.



When it has enough random information, it will create your key.



Now you should change the `Key comment`, e.g. something as fed4fire-youraccount, and provide a passphrase on your SSH key. After that, click `Save private key` and save it to a file on your PC, e.g. `fed4fire_youraccount.ppk`. Copy also the public key on top (in the frame below `Public key for pasting into OpenSSH authorized_keys file`) and save it to a local file, e.g. `fed4fire_youraccount.pub`. This is NOT the same format as the button `Save public key` will generate !



UNIX platforms (Linux/Apple OS X)

For UNIX platforms, creating an SSH key can be done through a command line tool that you run in a terminal. Run the command `ssh-keygen -t rsa`. The default file locations are normally okay, so you can press enter to accept them. Provide a passphrase on the SSH key.

Afterwards you have a file with the public part of your key (e.g. `id_rsa.pub`) and a file with the private part of your key (e.g. `id_rsa`). The latter may never be distributed.

Output looks like:

```
ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/home/bvermeul2/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/bvermeul2/.ssh/id_rsa.
Your public key has been saved in /home/bvermeul2/.ssh/id_rsa.pub.
The key fingerprint is:
4b:da:d0:55:4c:1b:fd:14:e6:6f:dd:79:c9:14:26:b0
bvermeul2@node0.singlenodedeb64.bvermeul.wall1.ilabt.iminds.be
The key's randomart image is:
+--[ RSA 2048]----+
|            +=o =.|
|            .oo* o|
|           .E.  = |
|        . .    o B|
|       . S      +*|
|        = .     ..|
|       . o        |
|                  |
|                  |
+-----------------+
```

Appendix J: Run a First Experiment

Introduction

The remainder of this appendix describes how one can run an experiment in real life on the federation (using jFed). This appendix is intended as an illustration of the architectural concepts described in this deliverable, to make them as clear as possible. The actual content of this appendix can be situated somewhere between architectural descriptions and more detailed specifications. Therefore it is likely that this information (possibly in more detail) will also become part of WP5's specifications for cycle 2 in deliverable D5.2.

When you have a local Fed4FIRE certificate (see ***Get An Account and Certificate***) and created an SSH key (see ***Create SSH key***), we can go on with a first experiment to see if you can access testbed resources. For this we will use the jFed tool.

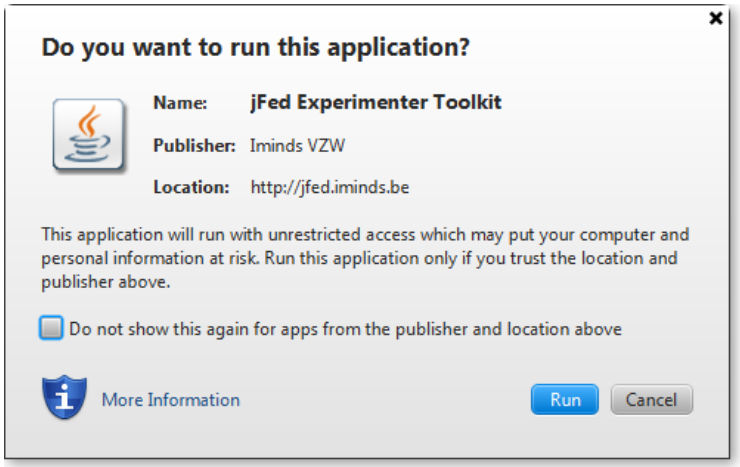
Start up jFed

You need to have a recent java 7 running. If not, please install from http://www.java.com.

Verify with http://www.java.com/verify that you run version 7.

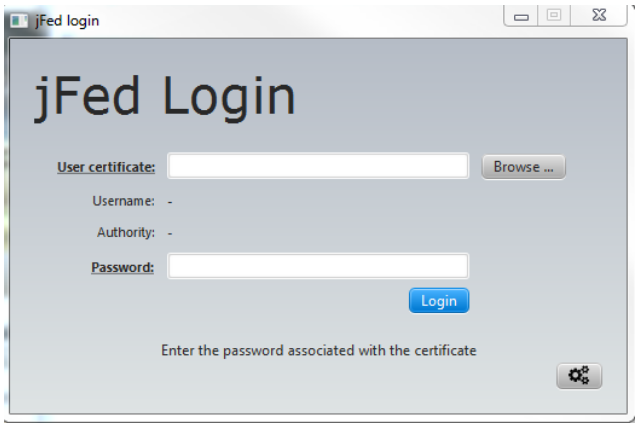
Go to http://jfed.iminds.be and click the green button `Quickstart jFed experimenter tool`.

A dialog box will pop up saying that the Java application is signed by iMinds. Click `Run` and optionally you can tick the box 'Do not show this again'.



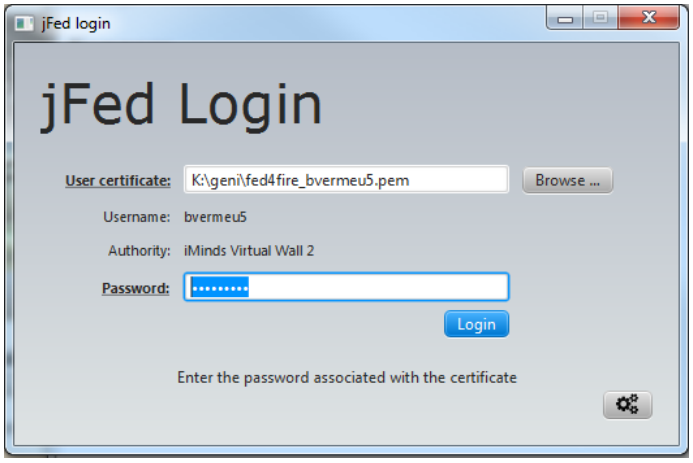
Run jFed for the first time

When jFed starts up for the first time, you will get the dialog box as shown below.

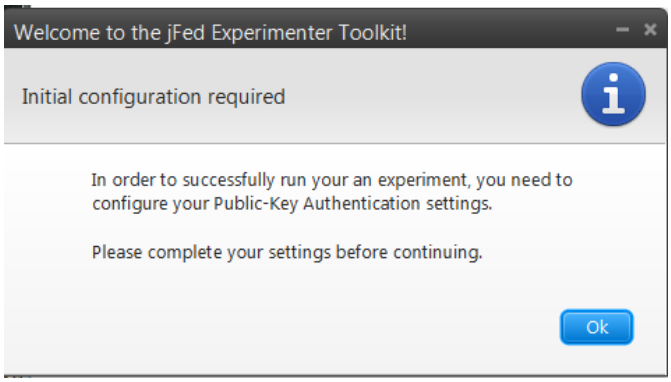


You should browse to the `pem` file that you downloaded from the authority portal. (see ***Download your Fed4FIRE certificate***).

Now jFed should automatically show your `Username` and `Authority` as shown below.

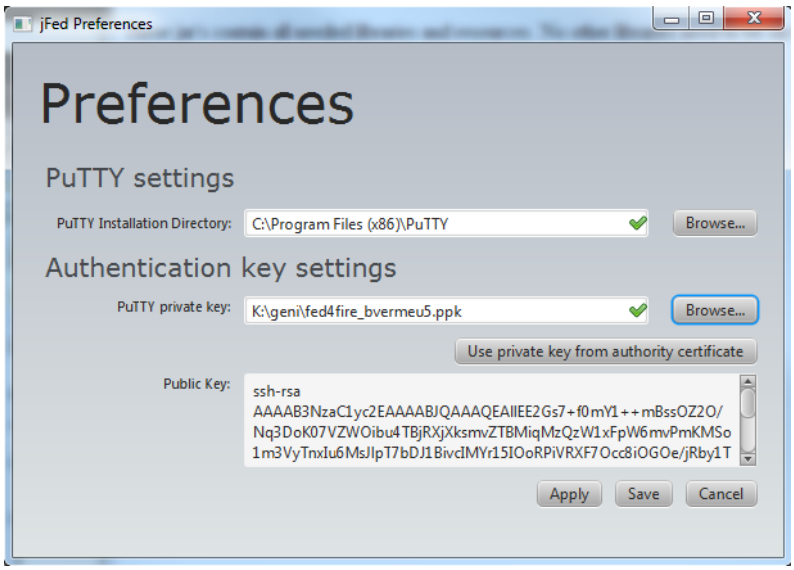


Type in your passphrase for this certificate and click `Login`. A dialog box will pop up to say that you have to configure jFed for this initial run.



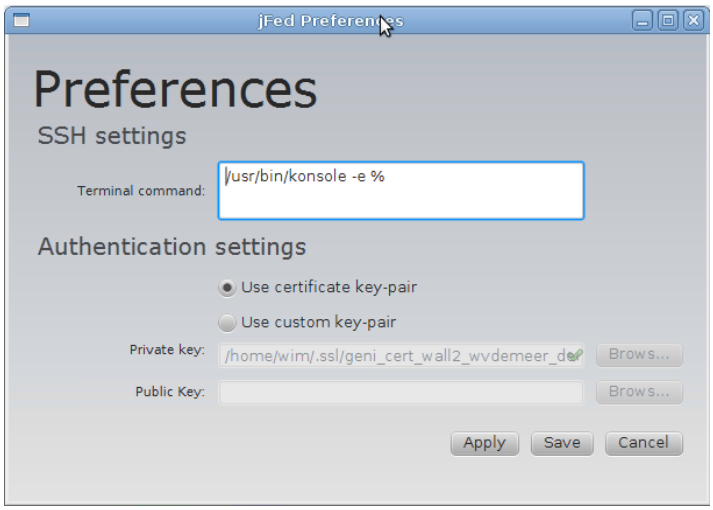
For Windows:

In this preferences settings you should point jFed to the PuTTY installation directory (see ***Windows platform: PuTTY***). Secondly, you should point jFed to the ***private SSH key*** (`ppk` file) you saved on your PC. Automagically, jFed should show the 'Public key' below. Click `Save` at the bottom right to save these settings.



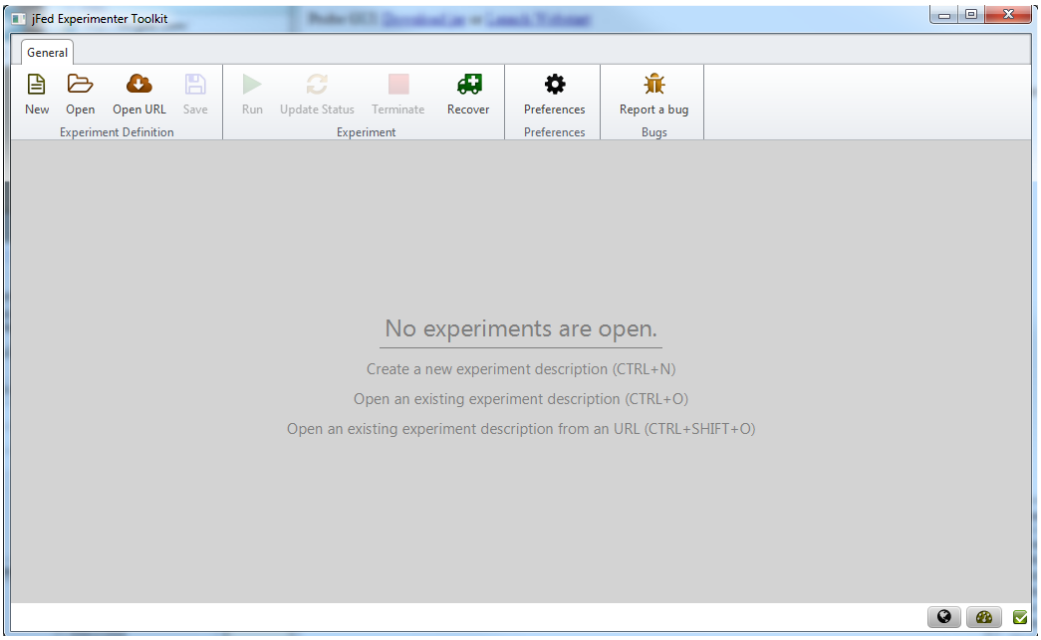
For Unix/Mac:

In this preferences settings, jFed should have a reasonable terminal configuration, so only change the default if it doesn't work when logging in. Secondly, you should click `Use custom key-pair` and point jFed to the ***private and public SSH key***you saved on your PC. Click `Save` at the bottom right to save these settings.

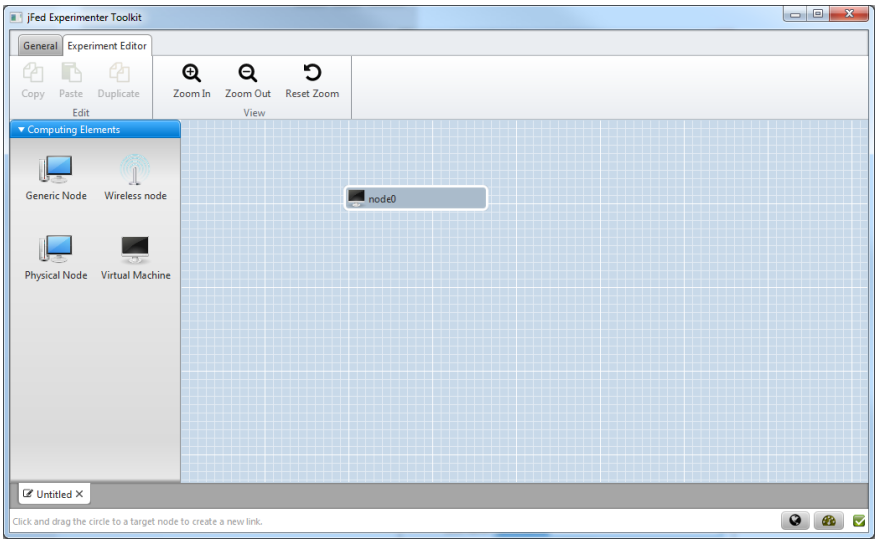


Create your first experiment

When you have logged in, and filled in your preferences, you see jFed with no experiments loaded.

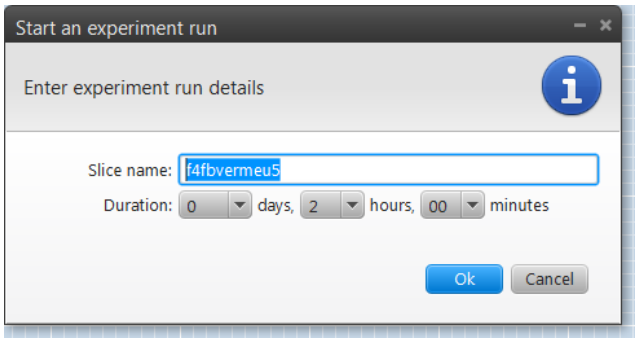


When you click `New`, you get a blank canvas where you can draw your experiment. Let's drag in a `Generic node` from the left side to the canvas. For more specific experiments you can right click and configure the node, but for now let it in the default settings.



Run the experiment

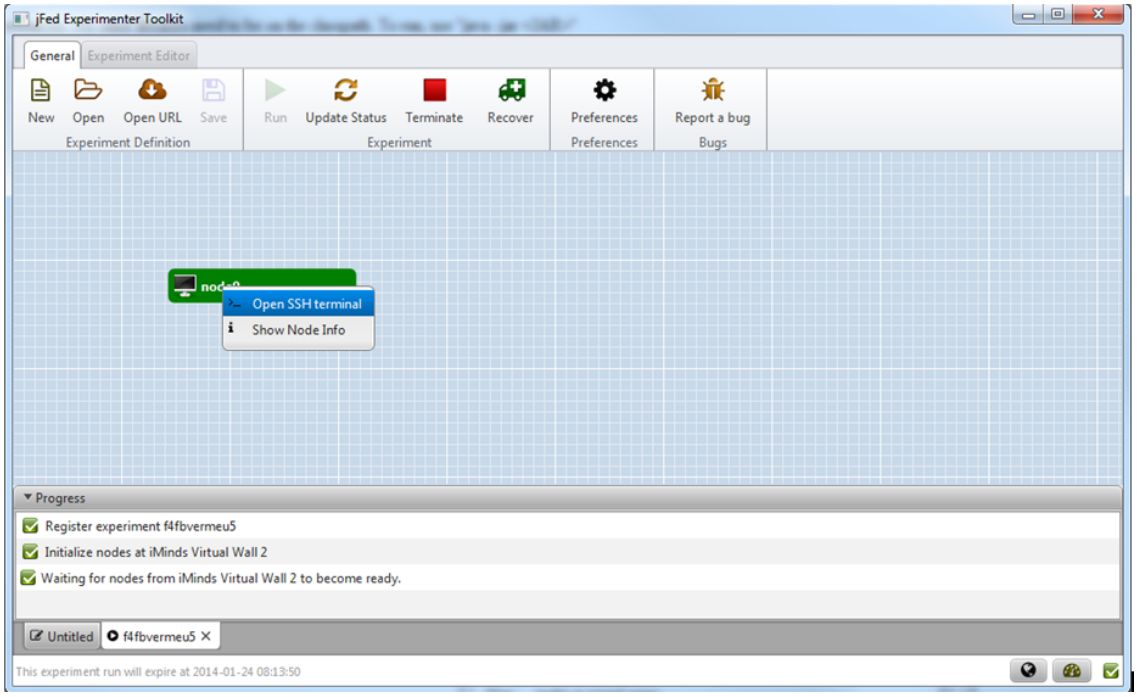
Let's run this experiment, by clicking the tab `General` at the top, and then the `Run` button. We will now have to choose a name for the experiment (= `slice name`) and choose a maximum duration.



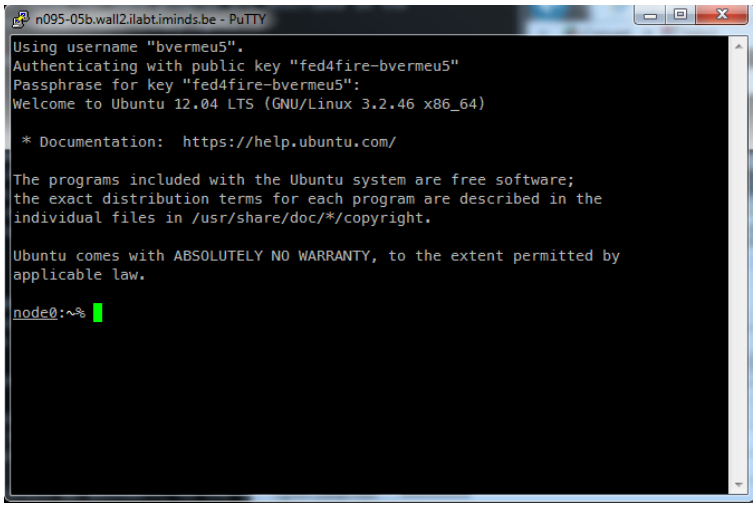
It will now take a couple of minutes to get the node prepared

Login on a node of the experiment

When the node becomes green, we can right click on the node, and click `Open SSH terminal`.

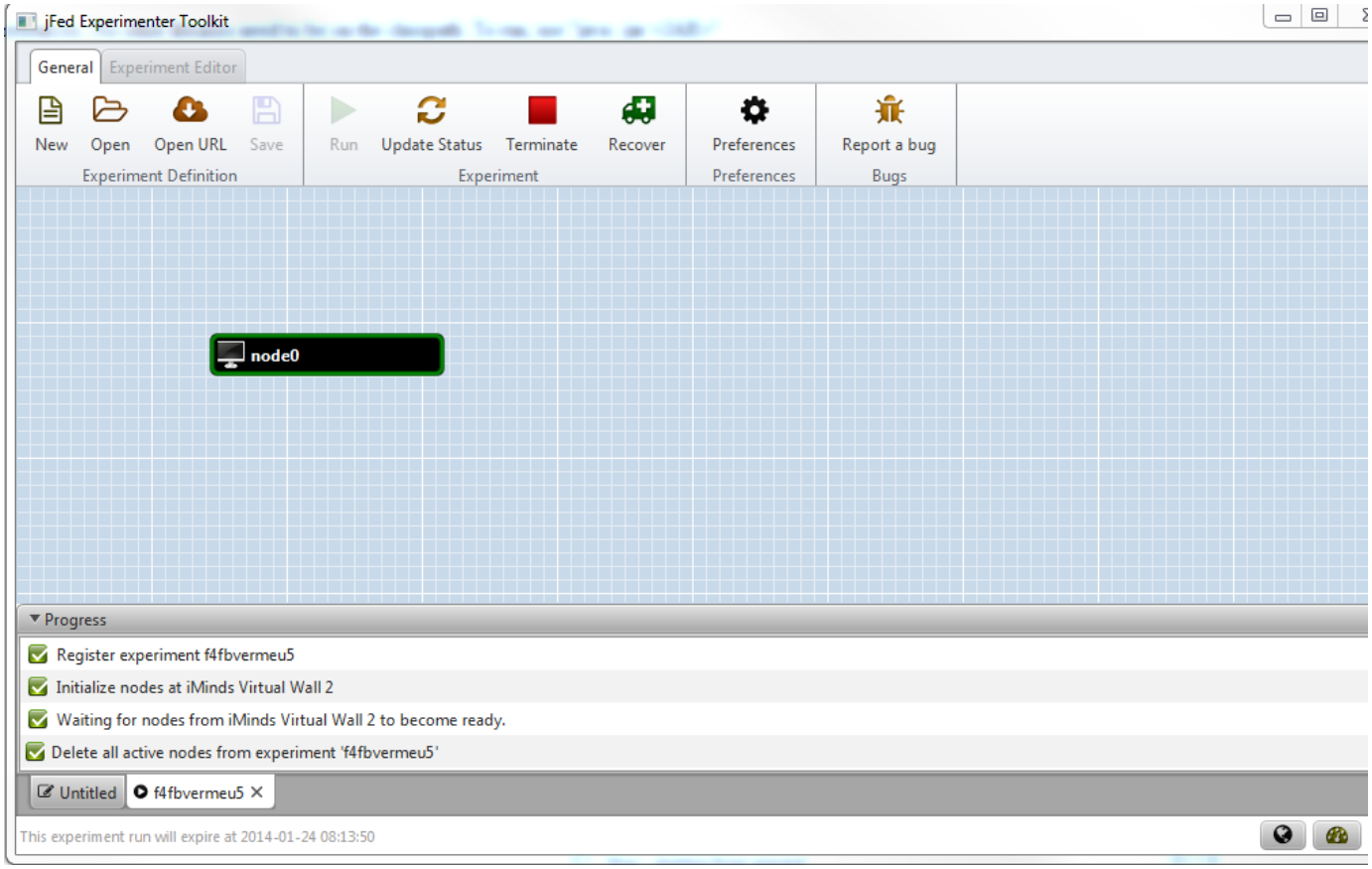


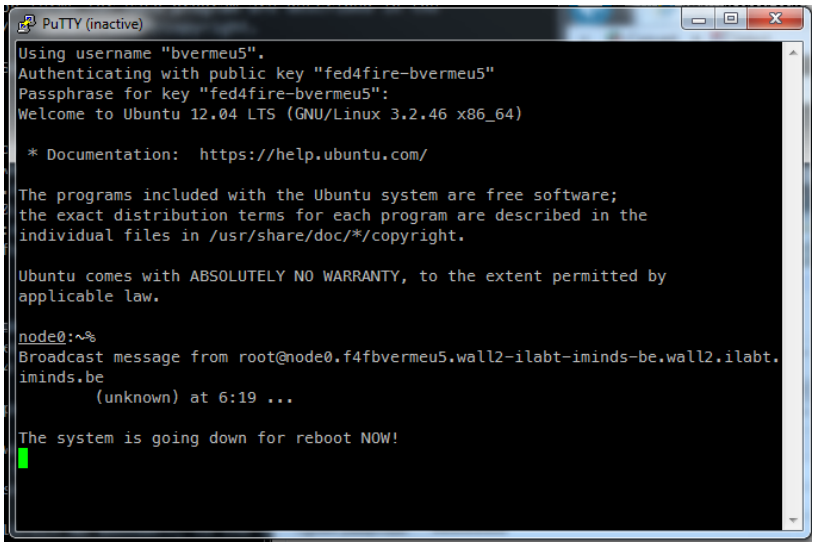
And then it will ask the passphrase on your ssh key. If the node says `Key refused` or another error it means something has gone wrong. See ***Note on connectivity***.



Ending the experiment

To release your resources before the endtime of your experiment, you can click the `Terminate` button at the top in jFed. After that the nodes will become black and if your ssh connection is still open, you can see that the node will be shutdown.





Note on connectivity

As in Europe public IPv4 addresses are scarce, we have the following problems for getting connected to the nodes:

- testbeds as Virtual Wall or w-iLab.t are only accessible through IPv6
- testbeds as BonFIRE have only a limited number of public IPv4 addresses, which is minimal in relation to the number of virtual machines they run.
- other testbeds run only on private IPv4 address and should be accessed through a gateway node.

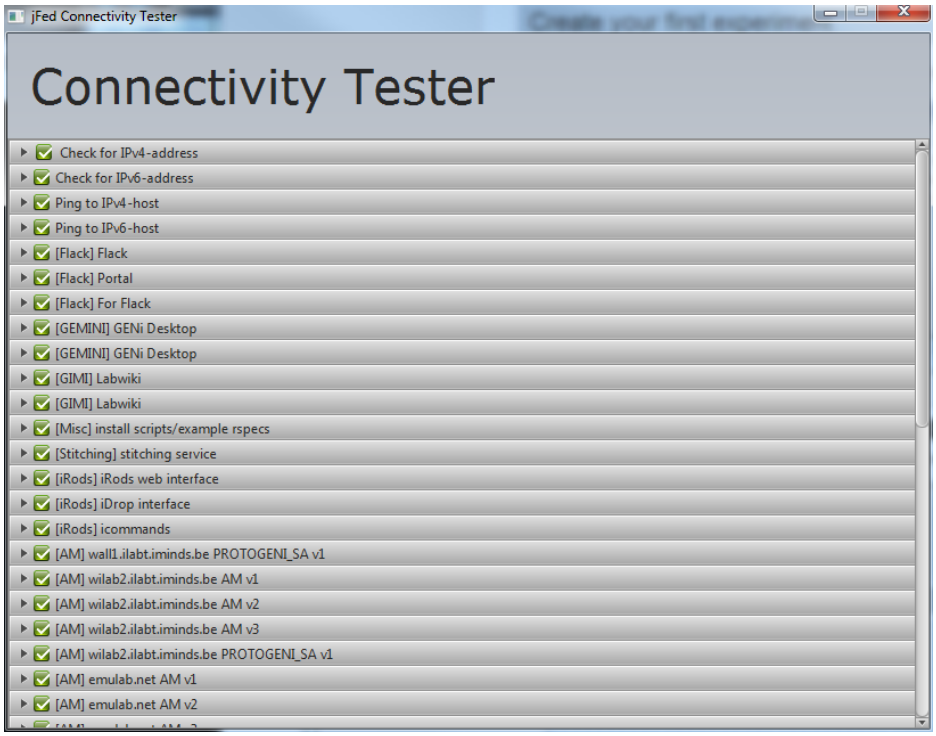
We are currently working around this in several ways, but for this first testrun now:

- the default node that was selected is at the Virtual Wall testbed (which is only accessible through IPv6).
- if you have IPv6 all will be okay and you will be able to login on the node.
- If the node becomes green, it means everything is allright with your account and certificate setup. You can stop the experiment now without login.
- if you don't have IPv6, register for an IPv6 address and tunnel, e.g. at Sixxs (choose AYIYA tunnel) and install Aiccu. This will take a couple of days to get approved, so `Terminate` this experiment with the button on top in jFed if you have one running.
- instead of a Generic node, you can draw a Virtual Machine on your canvas, and you will get a BonFIRE Virtual Machine with a public IPv4 address. for BonFIRE, you need an additional account (see ***BonFIRE***).

Test connectivity

You can test your connectivity with jFed by clicking the small button on the bottom:

And you will see a connectivity report:



Bug report with jFed

In case you cannot get a green node on your canvas (e.g. at the bottom of jFed you see problems passing by), click the `Report a bug` button in jFed and fill in a bug report. This will send all relevant information on calls and connectivity to jFed staff, so they can investigate the problem and report back to you.

The reporter email address is standard filled in with your certificate email address and this is forwarded by the authority to your own email address, so you don't have to change this.