Detailed specifications for first cycle ready

Florian Schreiner, Wim Vandenberghe, Loïc Baron, Carlos Bermudo, Albert Vico, Donatos Stavropoulos, Mohamed Amine Larabi, Lucia Guevgeozian Odizzio, Alina Quereilhac, Thierry Rakotoarivelo, et al.

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**

D5.1 Detailed specifications for first cycle ready

Work package	WP5
Task	Task 5.1
Due date	28/02/2013
Submission date	01/03/2013
Deliverable lead	Florian Schreiner (FOKUS)
Version	1.1
Authors	Wim Vandenberghe (iMinds) Loïc Baron (UPMC) Carlos Bermudo (i2CAT) Albert Vico (i2CAT) Donatos Stavropoulos (UTH) Mohamed Amine Larabi (INRIA) Lucia Guevgeozian Odizzio (INRIA) Alina Quereilhac (INRIA) Thierry Rakotoarivelo (NICTA) Chrysa Papagianni (NTUA) Georgios Androulidakis (NTUA) Aggelos Kapoukakis (NTUA) Florian Schreiner (FOKUS) Alexander Willner (TUB)

Reviewers	Max Ott (NICTA) Steve Taylor (IT-Innovations) Felicia Lobillo (Atos) Bernd Bochow (FOKUS)

Abstract	This document specifies the implementations and developments to be carried out in the context of Fed4FIRE's 1st development cycle focusing on WP5 "Experiment Lifecycle Management" mechanism and tools.
Keywords	Fed4FIRE Experiment Lifecycle Management, resource discovery, resource provisioning, resource reservation, experiment control, report

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

Fed4FIRE's "experiment lifecycle management" WP5 together with WP6 "monitoring and measurement" and WP7 "trustworthiness", represent Fed4FIRE's "federation-wide" mechanisms, i.e. unified mechanisms that are applied across heterogeneous testbeds and facilities.

This deliverable D5.1 is the first deliverable of WP5, describing the specifications for the first development cycle. As a fundamental basis, the specifications of the first development cycle in WP5 take into account and align as much as possible with D2.1 the "first federation architecture", taking also into account D3.1 "Infrastructure Community Federation Requirements" as well as requirements laid out by D4.1 the "First Input from Community to Architecture" and D8.1 "First Level Support".

Since the beginning of Fed4FIRE, significant efforts of WP5 went into the process of aggregating information, classifying, evaluating and selecting the most appropriate existing tools that have the potential to satisfy both architecture and community requirements and also integrate with the other federation-wide mechanisms from WP6 and WP7.

This document starts with describing the evaluation process leading to the selection of tools to be used and to be enhanced for WP5's first development cycle. The final outcome of this process is summarized in Table 1. Subsequently, this document describes the selected tools in detail, and provides a description of the exact functionalities expected to be ready after the first development cycle of WP5. Special emphasis is furthermore put on a description of the requirements that each tool brings to testbeds in order to interoperate.

Functional element of the Fed4FIRE architecture	**Implementation strategy**
Portal	Evolution of MySlice
Testbed directory	Extension of SFA API and MySlice Database, MySlice plugin
Tool directory	Extension of SFA API and MySlice Database, MySlice plugin, Wiki
Future reservation broker	Evolution of NITOS scheduler
Exposing testbeds through SFA	Initially evolution of SFAwrap, in the mid-term potential use of AMsoil
Experiment control	Cycle 1: FRCP and EC deployment on testbeds through OMF6 install. Cycle 2: add NEPI which interacts with the deployed FRCP layer.
Support of existing experimenter front-ends and tools	VCTTool and FCI, Flack, Omni, SFI

Table 1: Summary table mapping Fed4FIRE's architecture to selected solutions for cycle 1

Glossary

AA	Authorization and Authentication
AM	Aggregate Manager
AM	Aggregate Manager
API	Application Programming Interface
CLI	Command Line Interface
CMS	Content Management System
Fed4FIRE	Federation for Future Internet Research and Experimentation Facilities
FCI	Federation Computing Interface
FRCP	Federated Resource Control Protocol (FRCP)
GUI	Graphical User Interface
OCF	OFELIA Control Framework
OFELIA	OpenFlow in Europe: Linking Infrastructure and Applications
OMA	Open Mobile Alliance
OMF	cOntrol and Management Framework
PI	Principal Investigators
RA	Resource Adapter
REST	Representational State Transfer
RPC	Remote Procedure Call
RSpec	Resource Specification
RSpec	Resource Specification
SFA	Slice-based Federation Architecture
SSH	Secure Shell
UI	User Interface
URL	Uniform Resource Locator

VCT	Virtual Customer Testbed
VM	Virtual Machine
XMPP	eXtensible Messaging and Presence Protocol

Table of Contents

List of Figures

List of Tables

1 Introduction

Based on the cycle 1 architecture described in D2.1 [1], this document aims to provide detailed specifications of the targeted developments of Fed4FIRE's Experiment Lifecycle Management WP 5 for the first development cycle. Immediately after submission of this deliverable implementation work for Fed4FIRE's first development cycle will start so that after 7 months of implementation and alfa-testing work and two months of interoperability and beta-testing, the deployment of the described mechanisms on Fed4FIRE's various testbeds will be finished after project month 16 and ready to be utilized by Fed4FIRE's experimenters.
The specifications provided in this document cover all details needed to start the actual implementation work. If specific implementation choices defined in this deliverable result in a deviation of requirements as originally stated in D2.1 they are being identified and fed back to WP2 for further refinement of Fed4FIRE's overall architecture.

This document is structures as follows. The for WP5 relevant aspects of Fed4FIRE's initial architecture blue-prints of D2.1 are recapitulated in Section 2.1 as they represent the design boundaries according to which WP5 mechanisms have to be designed. Thereafter, requirements of Fed4FIRE's architecture, infrastructure community D3.1 [2] and services community D4.1 [3] are being mapped to the functionalities expected after Fed4FIRE's first development cycle in Section 2.2. Potential other requirements such as legal, contractual and governance requirements are provided in Section 2.3.
The actual specifications of Fed4FIRE's experiment lifecycle management building blocks are provided in Section 3. Each section specifying the six components (Portal 3.2, Testbed Directory 3.3, Tool Directory 3.4, Reservation Broker 3.5, Testbed Provisioning Interface 3.6 and Experiment Control 3.7) targeted for Fed4FIRE's first development cycle starts with a general description, evaluates possible approaches and available tools, describes the selected approach, details required additional implementations, describes the requirements for testbeds to interwork with and/or adopt a specific component and provides specifications regarding API, data and message formats and syntax.
Finally chapter 4 concludes this document mapping the planned components to the original architecture of D2.1 [1], describing deviations from the originally planned architecture and providing information about foreseen improvements.

2 Inputs to this deliverable

This section reviews some of the information provided by earlier Fed4FIRE deliverables. The goal of this exercise is to summarize the specific constraints that WP5 has to operate within when defining the specifications for the first cycle of Fed4FIRE.

2.1 Architecture

2.1.1 Resource discovery, resource requirement, resource reservation and resource provisioning

In Figure 1, the components of the architecture for cycle 1 of Fed4FIRE are depicted that play a role in the steps of resource discovery, resource reservation, resource reservation and resource provisioning of the experiment lifecycle. For a more in-depth discussion of these architectural elements we refer to D2.1 – First federation architecture [1].

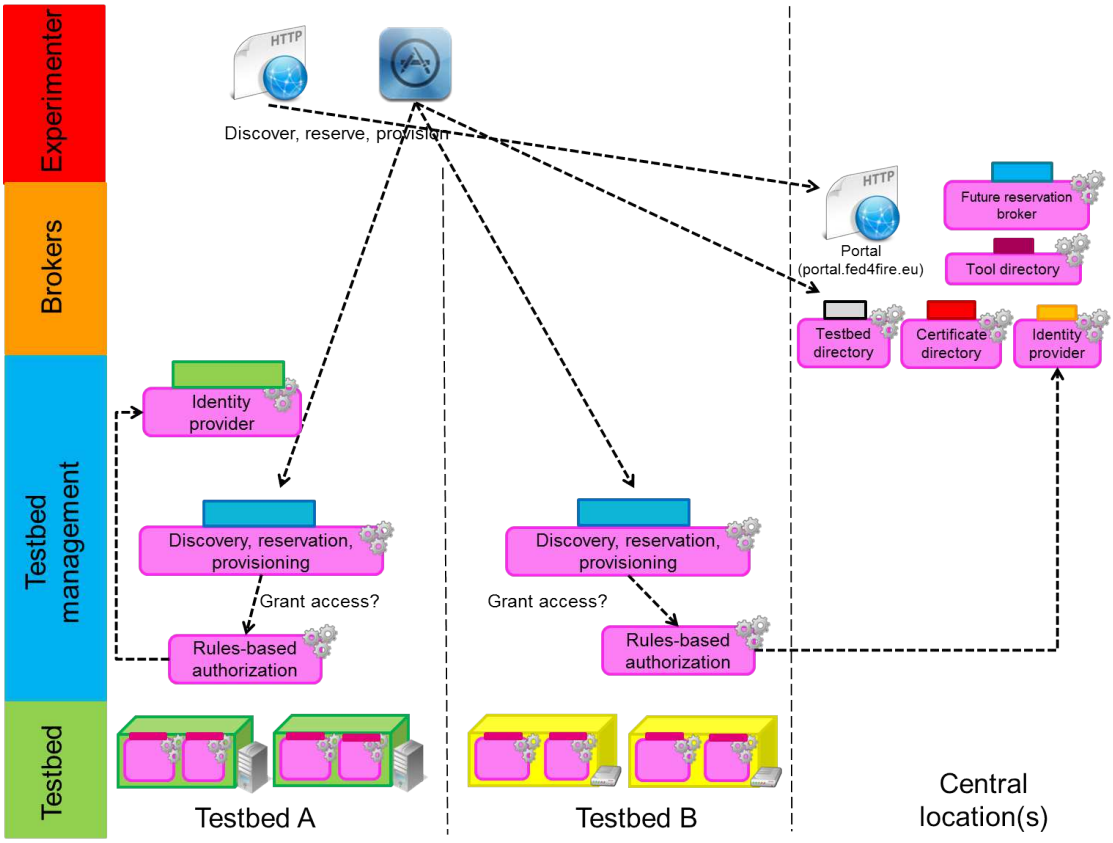

Figure 1: Fed4FIRE cycle 1 architecture for discovery, reservation and provisioning

2.1.2 Experiment control

For experiment control, the testbeds or central locations should not run specific components, as the experimenter can fully roll this out on his own. However testbed providers could ease this by putting certain frameworks pre-installed in certain images. Figure 2 shows two experiment

control frameworks each with their own interfaces and experimenter user tools/command line tools.

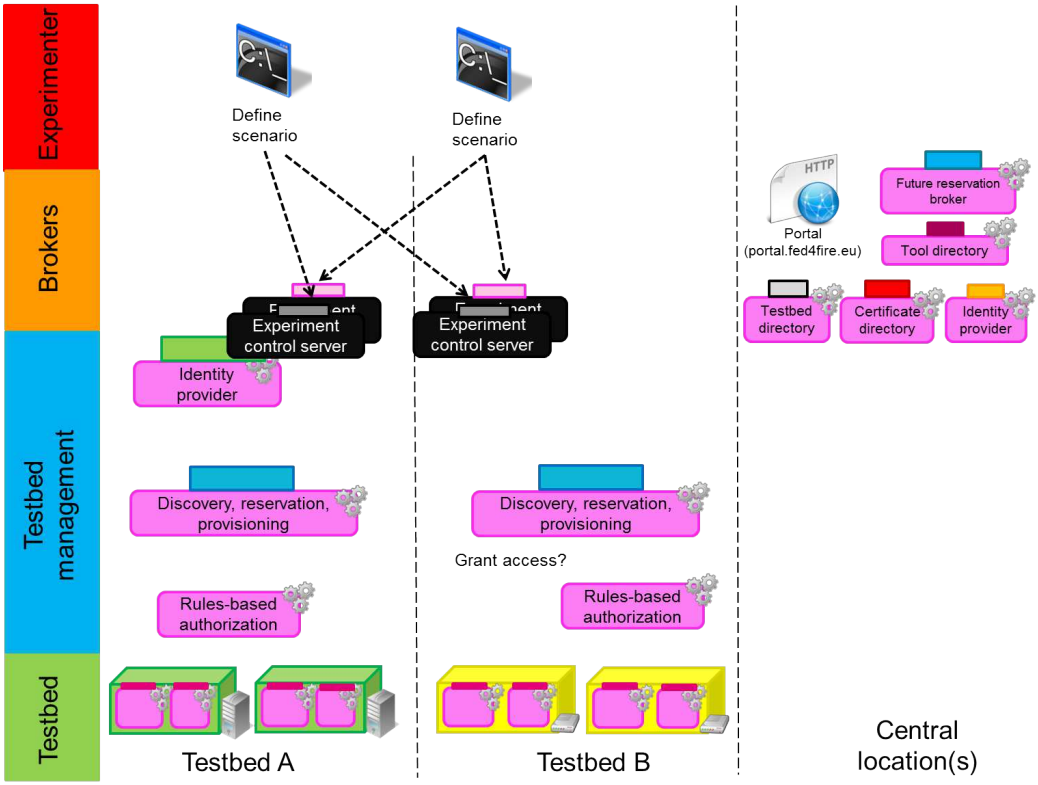

Figure 2: Cycle 1 architecture for Experiment Control

2.2 Requirements addressed by the architecture

This section presents a recapitulation of the requirements covered by the architecture described in D2.1 (section 5 of that document) [1]. This is a subset of all requirements written down in D3.1 [2] and D4.1 [3] and D8.1 [4], together with a number of other requirements introduced by D2.1. The intention is to recap only the requirements covered by the architecture which are relevant to WP5.

2.2.1 Generic requirements of a FIRE federation

The following section briefly recapitulates the requirements put forward by D2.1 with regards to scalability, support and ease of use for the experimenter. Some of them are expressed in the format of a questionnaire following the methodology used in that task (Q/A). Others are just requirements (R/A).

- **Scalability:**
  - Question: How can the architecture cope with a large number and a wide range of testbeds, resources, experimenters, experiments and tools?

- *Answer: As tools can speak directly to the testbeds through SFA in a peer-to-peer way, this is inherently scalable. The same applies for multiple identity providers with a chain of trust model. For very large experiments over multiple testbeds, experimenter tools can directly talk to all testbeds or broker services can orchestrate this (making the experimenter tool simpler).*

- **Support:**
  - Question: How easily can components/testbeds/software be upgraded?
  - *Answer: For this, the APIs should be versioned and tools and testbeds should support 2 or 3 versions at the same time, so that all components can be gradually upgraded.*
  - Question: How can different versions of protocols be supported? (e.g. upgrade of RSpec)
  - *Answer: With versions.*

- **Experimenter ease of use:**
  - Requirement: The final goal is to make it easier for experimenters to use all kinds of testbeds and tools. If an experimenter wants to access resources on multiple testbeds, this should be possible from a single experimenter tool environment.
  - *Answer: It is possible, but Fed4FIRE should also aim to keep such tools up-to-date during the lifetime of the project and set up a body which can further define the APIs, also after the project.*

2.2.2 Requirements from a sustainability point of view

The following section briefly recapitulates the requirements put forward by D2.1 with regards to Sustainability.

- Requirement: It is also required that the federation framework supports the joining and leaving of testbeds very easily, as this will be common practice.
  - *Answer: The architecture supports multiple identity providers/portals. There is a common API for discovery, requirements, reservation and provisioning while it imposes no restrictions on the use of specific experiment control, monitoring and storage. The common API makes it straight forward to add new tools and testbeds while a testbed can be an extra identity provider also.*

- Requirement: New tools can be easily added, while the dependency on specific tools or components for the federation should be minimized in order to avoid that the end of support of a specific tool makes the federation unusable
  - *Answer: See previous.*

- Requirement: Finally, it is also required that the experimenters can join and leave the federation easily, there is some notion of delegation (to make it more scalable

for lots of experimenters) and PIs (principal investigators) can put an end time on experimenters (e.g. students) or can withdraw experiments they have approved.

- *Answer: The architecture supports this through certificates but, of course, the certificate creation and sign up process itself has to be defined in detail in WP7* "trustworthiness" *and Task 5.6 (Portal definition).*

2.2.3 High priority requirements of the infrastructure community

The following Table 2 recapitulates the high priority requirements of the infrastructure community, as specified in D3.1. Requirements that will be met after cycle 1 are listed in "green", requirements that can be met partially are listed in "orange" and requirements that will not be met after cycle 1 are listed in "red" colour.

Federation aspect	Req. id	Req. statement	Remark
Resource discovery	I.1.101	Node capabilities	In cycle 1, the basic node capabilities can be retrieved in the same way. Not all facilities will provide all capabilities, but will provide the most tangible for their infrastructure. In cycle 2 and 3 this will be further refined to an ontology model of the resources.
Resource discovery	I.1.105	Discovery through federation-wide APIs	SFA will be adopted for this on all testbeds.
Resource discovery	I.1.106	Intra-infrastructure topology information	Will be provided as is by the facilities, but will be refined further in cycle 2 and 3 with the ontology model
Resource discovery	I.1.107	Inter-infrastructure topology information	Interconnecting infrastructures in a structured way is not targeted for cycle 1
Resource discovery	I.1.109	Query search	Based on the resource discovery of SFA, this will be possible. Brokers (with the same interface) can help in doing this on multiple testbeds at once.
Resource discovery	I.1.110	Catalogue search	Will be tackled by the portal which will line up all testbeds, and user tools which listen up all resources based on the discovery phase.

Resource requirements	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).
Resource reservation	I.1.301	Hard resource reservation	SFA will be extended to cope with future reservations. Testbeds should implement hard reservation (is not an architecture requirement).
Resource reservation	I.1.304	Automated reservations handling	SFA will be extended in this way, brokers can handle this in a multi-testbed way.
Resource reservation	I.1.305	Reservation information	SFA will be extended in this way
Resource provisioning	I.1.401	Provisioning API	Will be done by SFA
Resource provisioning	I.1.403	Root access	No architecture requirement
Resource provisioning	I.1.404	Internet access to software package repositories	No architecture requirement
Experiment control	I.1.501	SSH access	Architecture will make it possible to distribute SSH public keys as part of the SFA API (geni_users option in provision call).
Experiment control	I.1.502	Scripted control engine	Architecture facilitates the use of experiment control engines
Experiment control	I.1.504	Generality of control engine	No architecture requirement
Inter-connectivity	I.4.001	Layer 3 connectivity between testbeds	Not present in a structured way in cycle 1.
Inter-connectivity	I.4.003	Transparency	Not present in a structured way in cycle 1.

Inter-connectivity	I.4.005	IPv6 support	In cycle 1, some testbed resources will be reachable through IPv6. The architecture can cope with this, e.g. through DNS names which resolve to IPv6
Inter-connectivity	I.4.006	Information about testbed Interconnections	Not present in a structured way in cycle 1.

Table 2: High priority requirements of the infrastructure community (D3.1)

2.2.4 High priority requirements of the services community

The following Table 3 recapitulates the high priority requirements of the services community, as specified in D4.1. Requirements that will be met after cycle 1 are listed in "green", requirements that can be met partially are listed in "orange" and requirements that will not be met after cycle 1 are listed in "red" colour.

Field	Req. id	Req. statement	Remark
Resource Discovery	ST.1.005	Describe connectivity options	Interconnectivity in a structured way is not tackled in cycle 1
Resource Provisioning	ST.1.007	Experiment Deployment	Architecture makes it possible to deploy such tools.
Resource Requirements	ST.1.013	Resource description	Identical to I.1.101
Resource Reservation	ST.1.017	Testbed reservation information and methods according to experimenter profiles/policies	Similar to I.3.201
Resource Reservation	ST.1.020	Scheduling parallel processing	The architecture in cycle 1 can cope with this, but interconnectivity will not be supported yet in a structured way.
Experiment Control	ST.1.023	Access to internal and external services from a federated testbed	Not an architecture requirement.
Experiment Control	ST.1.029	Multiple testbeds available	The architecture in cycle 1 can cope with this, but interconnectivity will not be

			supported yet in a structured way. Some testbeds will probably use public IP addresses (IPv4 or IPv6) which can be used for inter testbed communication over the public internet.
Authorization	ST.3.002	Single sign on for all testbeds required for experiment	This is tackled by using certificates signed by identity providers. Once you upload this certificated in the experimenter tool and provide a passphrase, you can use it on all testbeds.
Interconnectivity	ST.4.001	Access between testbeds (interconnectivity)	Interconnectivity in a structured way is not tackled in cycle 1

Table 3: High priority requirements of the services community (D4.1)

2.3 Additional WP5 requirements: Legal, contractual aspects, governance requirements

Until now, only WP3 “Infrastructure”, WP4 “Services and Applications “ and WP8 “Shared Support Services” have submitted deliverables that define their specific requirements towards the Fed4FIRE federation framework. Technical requirements in order to adopt the functionalities specified in this deliverable are summarized at the end of the specifications of each mechanism. In terms of legal, contractual aspects or governance requirements there were no additional requirements identified at the current stage.

3 Implementation of the architectural functional elements

3.1 Introduction

In this section we discuss all functional elements of the architecture related to WP5 “Experiment Lifecycle Management” and define how they will be implemented. In many cases, available software, tools or a combination of such software will be used as a starting point (in order not to reinvent the wheel, reduce implementation efforts, utilize already mature components and/or support already existing communities). However, some elements will be implemented from scratch. In the following sections, the tools / mechanisms are described, required implementations outlined and a formal definition of the corresponding APIs and data formats is provided.

Tools have been evaluated according to how much they provide the expected features, but also taking into account some assumptions that arise from the general approach Fed4FIRE is following, i.e., to try to take as much advantage as possible from existing tools and mechanisms in the facilities to be federated and to allow current experimenters of these facilities to keep on using their own tools with a broader range of resources. This approach leads to several assumptions, such as the use of SFA for resource exposure. This decision has conditioned the evaluation of the tools described in the following subsections and it is reflected as such.

3.2 Portal

3.2.1 General description

An increasing number of computer networking testbeds today are joining a federation based on the Slice-Based Federation Architecture (SFA) [1] (see also section 3.6) . SFA provides a 'thin waist' for secure, distributed resource requests. The same approach is adopted in Fed4FIRE, as has already been presented previously in the architectural deliverable D2.1 [1].

This effort has fostered the emergence of an ecosystem of tools and services covering a wide range of applications not included in the thin waist of SFA: enriched user interfaces, measurement and monitoring platforms, user registration and authentication services, and experimental management software, to name a few. However, a typical user willing to run an experiment across multiple testing facilities is usually required to switch between many heterogeneous and overlapping tools. He needs to discover them, learn their processes and semantics, and often authenticate himself several times and manually make the bridge between them. As a response to this diversity and complexity, the notion of an experimenter's portal is often advanced. To the best of our knowledge, it is fair to say though that the research community has so far not proposed a satisfactory solution that goes beyond sparse integration efforts.

The aim is to fill this gap and provide a portal, which will be a central starting point to access the Fed4FIRE federation. The portal will provide pointers to the project website, to first line support (WP8), to the federated testbeds' websites and to appropriate FIRE tools. Portal users will also be guided to the First Level Support systems (e.g. Trouble Ticket System), if they need help to register or use the portal or encounter problems while setting up experiments.

As described in D2.1, the portal will also be the registration place for new experimenters [1]. Therefore, it will provide an easy way for experimenters to register themselves and to access the federated testbeds. Note however that the testbeds will always determine whether the user can actually access them according to their access policies.

Moreover, the portal will also perform the role of a client tool. On behalf of the experimenter, the portal will forward queries to federated testbeds using SFA delegation mechanism. Using the portal, the experimenter will be able to search, browse and reserve resources across federated testbeds. The portal will also act as a bridge to experiment control tools.

It is obvious that the portal will provide a vast amount of functionalities. Many of them will be implemented in the portal itself, but for some tasks interactions with other functional elements of the Fed4FIRE architecture will be required. To be more concrete, the interactions between the portal and the other functional elements defined in the architecture are depicted in Figure 3. The portal includes some of the functionality of the testbed directory (the human readable part, see Section 3.3 for more details regarding the testbed directory specifications). It also includes the functionality of the tool directory (see Section 3.4). Both components will retrieve their needed information directly from the testbeds using the SFA interface. For user authentication, the portal will contact the identity provider of the federation if the user is not affiliated with a specific

testbed. If the user has such an affiliation then the portal will use the certificate directory to authenticate the user with the correct certificate. Finally, the portal will provide a GUI to reserve resources within the federation. For this it will interact with the future reservation broker (described in Section 3.5).

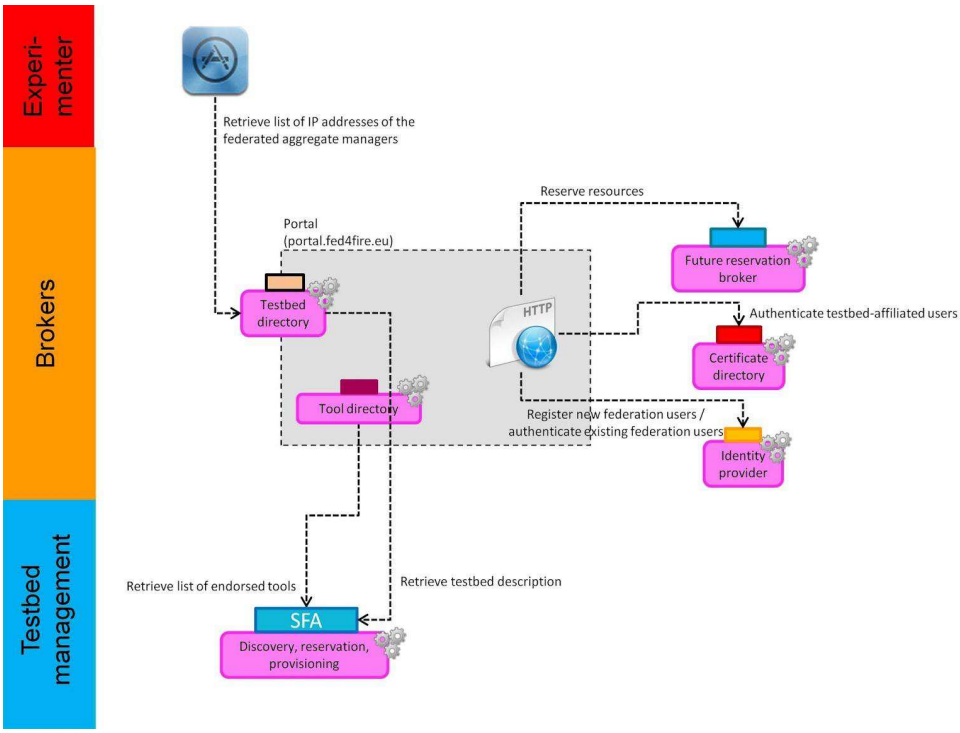

Figure 3: Portal interactions with external systems

3.2.2 Evaluation of possible approaches for implementation

This section, based on the requirements that were deducted by analyzing the high level architecture of Fed4FIRE (D2.1), analyses whether existing and available tools meet these requirements in order to identify and finally select the best suitable available solution as the basis for further developments.

Requirements:
Several tools have been evaluated to build the Fed4FIRE Portal according to the gathered requirements.

- In order to ensure the accessibility and the visibility of the federation, the first requirement for the portal is to have a web interface.
- As an experimenter's tool, the portal requires a programmatic interface such as an API, which allows automating user's actions.

- The portal is required to communicate with a central identity provider to register, authenticate and manage users. Thus, the portal will be a delegate of the user. It is acting on behalf of the user making the request.

- The portal has to ensure a chain of trust and act as a broker between users and testbeds (define WP7 "trustworthiness"). Through a single sign-on mechanism the portal enables the possibility for users to access all federated testbeds. Authorization is still under the responsibility of the testbeds, which have to ensure that incoming users are following their local policy rules.

- Since the Fed4FIRE architecture uses SFA as a core mechanism, the portal needs to be SFA compliant. The portal needs to make it straightforward to add new tools and testbeds. It needs to be able to display the diversity of testbeds' resources, such as geographical, topological, or 3D coordinates information.

- The portal has to provide an entry point for the following functionalities: discovery, reservation and provisioning of testbed resources. It should not impose any restrictions on the use of specific experiment control, monitoring and storage mechanisms.

- The portal represents *one* centralized and convenient way to access federated facilities but there might be other (sometimes preferred by experimenters) ways to interface with the federated testbeds. Therefore the federation as a whole should not depend and rely on the portal's services.

- Of course, the central portal, identity provider, testbed directory, tool directory and certificate directory ease the use of the federation as all the information is provided in a single place to get new experimenters use testbeds and tools.

How available and potentially eligible tools meet the requirements:

The following tools and software solutions have been analysed in terms of suitability for meeting the requirements arisen be Fed4FIRE's high level architecture. Table 4 gathers a summarized comparison of the following tools.

FITeagle's Virtual Customer Testbed (VCT) tool [6] is a standalone tool that can be used to discover, reserve and provision resources belonging to different SFA compliant testbeds. It can be considered as being the frontend of the FITeagle framework. This is an extensible and distributed open source experimentation and management framework for federated Future Internet testbeds. However, the VCT tool doesn't provide authentication and user registration management mechanisms. Moreover, it is Java-based software and does not correspond to the need of a web portal.

BonFIRE's portal [11] is a web portal that has been developed in the Bonfire project. BonFIRE's portal covers the main functionalities of discovery, reservation and resource provisioning. But it

doesn't support SFA, which is adopted in the Fed4FIRE federation architecture. One can notice another disadvantage, which is the lack of modular approach as there are no plugins to enhance the diversity of testbeds' resources.

Flack is a piece of software developed in the GENI project (USA) [12]. Flack covers the main functionalities of authentication, discovery and resource provisioning over SFA compliant testbeds. However, it doesn't provide user registration management as the GENI Clearinghouse covers this functionality. One can notice another disadvantage, which is the lack of modular approach as there are no plugins and only a map view of resources. Moreover, Flack is Flash-based software that does not correspond to the need of a web portal.

NEPI [8] **and LabWiki** [9] are more focused on experiment control than on resources discovery. Thus these tools cannot be selected as a basis for developing Fed4FIRE's portal, as resource discovery is a key feature and requirement for the portal. And although these tools might be complementary, providing resource control during experiment, this is not in the scope to the portal's functionalities.

MySlice [14] provides a user authentication and management system based on SFA. Therefore, it is able to use SFA Registry as a central identity provider for the federation. It provides the main functionalities for authentication, discovery, reservation and resource provisioning over several testbeds. It has a gateway system that makes it straightforward to add new tools and testbeds. MySlice is able to act as a broker to contact multiple testbeds and measurement sources that are aggregated in order to provide a consistent layout to the user. It provides both a Web interface and an API. It has plugins developed to fit the needs of a wide diversity of testbeds. It already has an active community of developers including partners of Fed4FIRE project. For all these reasons, MySlice has been selected as portal for Fed4FIRE's first architecture iteration cycle.

Approach	Advantages	Disadvantages	Selected as final approach
MySlice [14]	• Active community of developers and users • Only inclusion of additional methods needed, no changes to existing code required • Reasonable implementation effort needed • Provides both web UI and API • Modular design with plugins & gateways • Act as a proxy to contact multiple testbeds and measurement sources, is able to aggregate content		X

Approach	Advantages	Disadvantages	Selected as final approach
	• SFA compliant • User management • Authentication • Discovery • Reservation • Resource provisioning		
Lab Wiki [13]	• User friendly web UI	• Focused on Experiment Control	
NEPI [8]	• User friendly UI • API	• No web UI • Focused on Experiment Control	
FiTeagle VCT [6]	• User friendly UI • REST API • Resources description • Experiment description • Authentication • Discovery • Resource provisioning • SFA compliant	• No web UI	
BonFIRE Portal [11]	• Provides both web UI and API • Resources description • Experiment description • Authentication • Discovery • Rservation • Resource provisioning	• Not SFA compliant • No modular approach, i.e. implementation effort needed beyond adding modules	
Flack [12]	• Authentication • Discovery • Resource provisioning • SFA compliant	• Flash Web UI • No modular approach, i.e. implementation effort needed beyond adding modules No user management	

Approach	Advantages	Disadvantages	Selected as final approach
		• No API	
Build from scratch		• Total amount of needed implementation effort exceeds the available manpower by far.	

Table 4: Comparison of available tools for implementing Fed4FIRE's portal

3.2.3 Description of selected tool

From the previous evaluation, MySlice seems to best fit the needs of the Fed4FIRE requirements to provide access to the federation though a portal. In the following, MySlice is evaluated more deeply regarding specific Fed4FIRE operational and usability requirements. However, as described in section 3.2.4, several modifications of MySlice are to be met in order to provide all the features that Fed4FIRE federation requires.

Open platform approach

MySlice follows an open platform approach [17], which consists of a thin layer that provides the glue between existing components. MySlice [14] is a running prototype offering a single and consistent interface for authentication, resource browsing/selection, and measurements. An open community model invites developers to contribute and share extensions.

MySlice as a portal

MySlice sits at the edge of the federation and aims to be a trusted entity facing a community of experimenters. This privileged situation makes it possible to simplify access and enhance the portal user's experience, for instance, by handling complex procedures on his behalf, or pre-processing and caching data. It acts as an intelligent mediator that makes the bridge between the experimenter and the set of (generally distributed) available components. MySlice aims to hide some complexity of SFA from the experimenters/users. MySlice as a portal includes several functionalities such as user registration, account management and authentication, managing credentials on behalf of a user. An experimenter registered in MySlice is identified through a local user account in the MySlice database and references to his credentials. Experimenter's credentials are obtained from a platform known by MySlice. This platform needs to trust MySlice to act on behalf of its users. In Fed4Fire, a testbed's (local) SFA Registry or a central one can provide credentials (WP2 Architecture [1]).

MySlice is only one centralized and convenient way to access a federation of testbeds. There might be other mechanisms, e.g. direct utilization of APIs in a programmatic fashion, or other

tools more suitable to specific experimenters. Thus the availability of the federation as a whole doesn't depend on MySlice, the portal respectively. A failure or outage of the portal does not result in complete unavailability of the entire testbed federation (no single point of failure).

MySlice as a client tool

MySlice as a client tool provides a single tool to experimenters, which works across different platforms. It allows users to search, browse and reserve resources through a command line interface or a Web based graphical user interface. A novice user can therefore easily use the web UI. However, a more advanced user would be able to query the more powerful command line interface.

Combine multiple sources of data

One of MySlice's added values stems from of its capability to combine multiple data sources and services. The result in some cases provides information beyond the testbed itself.

MySlice provides a generic data description model, simple semantics for descriptions, and a lightweight communication protocol. A common abstraction to help users browse through and interact with a large amount of data sources is challenging. MySlice was built on the Manifold interconnection framework [16]. The current prototype features both the SFA and TopHat [18] measurement gateways, which are combined to augment the value for users. New platform APIs can be connected through a system of gateways. MySlice uses these various gateways to query different platforms using a common semantic.

As shown in Figure 4, experimenters can submit queries that get optimally and transparently dispatched to the different services capable of formulating an answer.

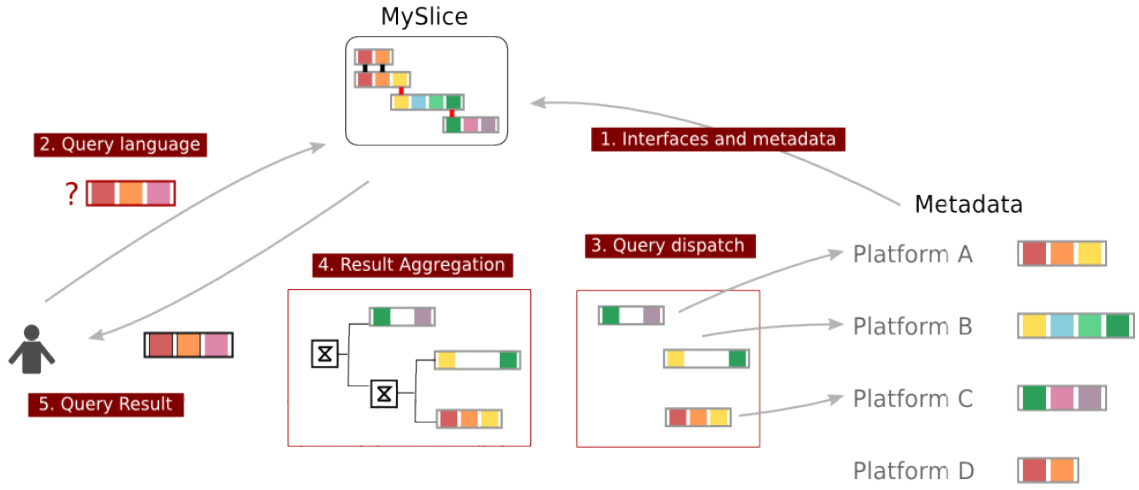

Figure 4: MySlice Gateways for dispatching queries across multiple platforms

Figure 4 also shows five of the most important interactions between MySlice, experimenters and platforms:

1) MySlice is able to contact several platforms and to gather metadata about the information provided by these.
2) User submits a query to MySlice either through the web interface or through the API.
3) The user's query is optimally and transparently dispatched to the different platforms.
4) The answers provided by the different platforms are aggregated as a single result.
5) The user receives a result to his query.

Expose the richness of heterogeneous testbeds

Another important capability of MySlice is its ability to adapt to resources and services specificities through a plug-in system, achieving a proper balance between the needs to present uniform interfaces and to expose the full richness of heterogeneous environments. MySlice is able to adapt to various data properties through this plugin system. Indeed, new plugins have been developed (more info on MySlice's Developer Website [22], including MySlice Plugin Developer Guide [19]) to fulfil specific needs such as representing testbed resources on a map (if latitude and longitude properties are available) or a 3D representation (if x,y,z coordinates are provided by the testbed) or a scheduling table (if scheduling information is provided for specific resources).

Through its current web interface MySlice provides resource visualization, filtering, selection and reservation functions that assist an experimenter in a structured analysis and interpretation of data. The reservation plugin is based on the joint work between Nitos and PlanetLab. This various functionalities are achieved through a plug-in system that enables a unified access to heterogeneous data sources, balancing the needs to present uniform interfaces and taking advantage of any heterogeneity of testbeds.

Architecture

The MySlice architecture is modular and flexible. It is composed of three parts: Core, API and Web. The architecture of MySlice, as depicted in Figure 5, allows testbed owners to design gateways beneath the core library and plug-ins on top of the web framework.

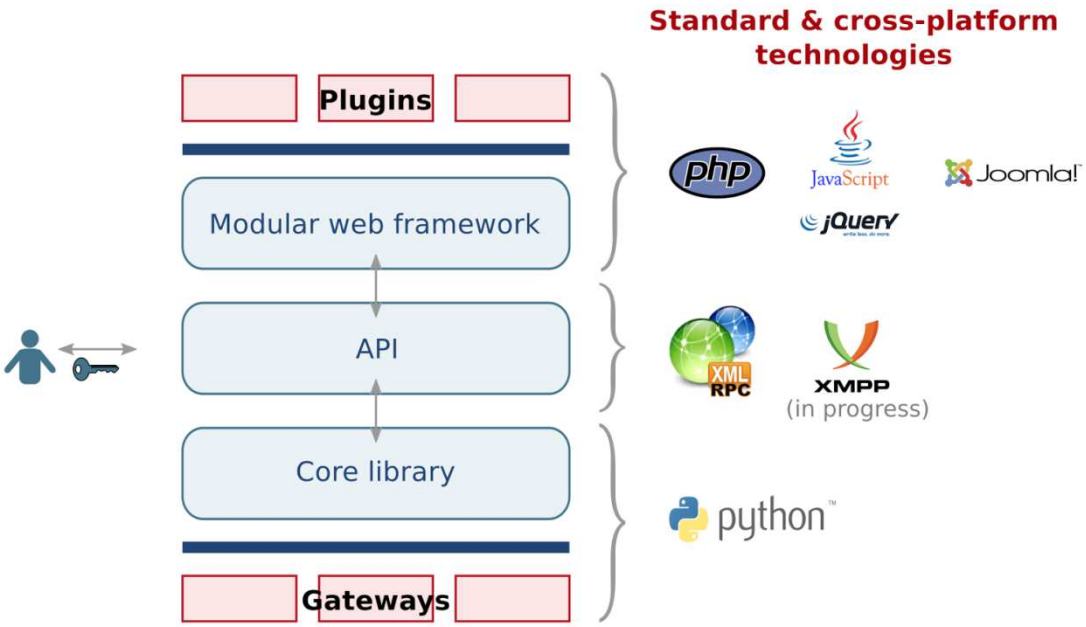

Figure 5: MySlice Architecture

Conclusion

Building a portal for testbed federation has long been a desired, but challenging enterprise. We propose an open and running platform based on the combination of existing and proven components. It provides the necessary glue between various components of interest to ensure their interoperability and proposes a consistent interface supporting, through extensions, the full experiment lifecycle. This architecture also encourages developers to build custom plugins and gateways specific to their needs. As an open platform, MySlice has a growing community of contributors. The required modifications of MySlice are described in the following subsection.

3.2.4 Required additional implementations

Table 5 describes planned enhancements of MySlice implemented in cycle 1 and beyond.

Functionality	Cycle 1	Further Cycles
Content MySlice is using Joomla [20]. This CMS allows easily to add content to the portal.	• Identify what contents, links and tutorials are most required and suitable (collaboration with WP9 “dissemination”). • Guide users to the First Level Support (WP8). • Develop the testbed and tools	

Functionality	Cycle 1	Further Cycles
	directories within the portal as specified in section 3.3 and 3.4.	
Registration The current status of the registration functionality is as follows: a user can register to a testbed (PlanetLab, OneLab...), a MySlice admin user can create a local account on MySlice through its command line interface.	• Create an SFA registry (central identity provider) for Fed4FIRE federation that will allow users that do not already have credentials at testbeds to connect to the portal • If a user already has a testbed credential, then testbeds should issue SFA compliant certificates, which can be used in the federation (D7.1). • Develop adequate plugins for MySlice to offer a basic registration functionality on the portal.	• Further develop the adequate plugins for MySlice to offer a convenient access for users and improve the registration functionality
Authentication Authentication to MySlice web UI is handled through login and password. A decision has to be made in WP7 ”trustworthiness” to choose the login process.	• Handle one login process according to WP7 requirements (OpenID, Mozilla Persona, SFA certificate, ...) • Generate key pairs on the portal, to avoid the bootstrap process through a script • Automatic delegation if the portal generated a private key for the user during registration	• Handle additional login processes
Authorization & Access policies Up to the testbed	• Integration of testbeds in MySlice • Manually approving users on the Fed4FIRE registry will grant access to the portal and to testbeds that agreed for an immediate access of users.	• Ask users to fulfil specific testbed’s policies, for example providing required personal information • Allow Testbed’s PI or Admins to validate or disable Fed4FIRE user

Functionality	Cycle 1	Further Cycles
		accounts
Testbed resources Relies on SFA mechanisms	• Allow users to browse, filter, add or remove resources in a slice and to create and update a slice across different testbeds of Fed4FIRE's federation. • Develop plugin that provides the functionality to reserve resources, as specified in section 3.5. • Additional testbeds will be plugged into MySlice	• Some partners will develop plugins to expose specific resources of their testbeds (e.g. OpenFlow resources) • These new plugins will have to be extended to be generic and flexible enough to be adopted by other partners • Integration and handover between MySlice and experiment control tools such as NEPI and OMF will be developed

Table 5: MySlice enhancement in 1st cycle and further evolution

3.2.5 Specifications

Content

On the portal, background information will be provided to the experimenters in the context of the Fed4FIRE federation. First of all, the portal will offer information regarding the testbeds that belong to the Fed4FIRE federation. Links to the project's website and training material will be put on the portal. The portal will also display a catalogue-like view on the testbeds through a specific plugin that implements the human readable version of the testbed directory as specified in section 3.3. The portal will also gather information regarding the different tools that can be used within the Fed4FIRE federation through a specific plugin that implements the tools directory. As specified in section 3.4, this directory is integrated into the portal. It will be based on Wiki technology. Finally, the portal will also provide assistance to the experimenters by guiding them towards the Fed4FIRE First Level Support service which is developed within WP8.

Registration

Users will be able to register to the Fed4FIRE federation. This first functionality will allow experimenters registration, but user accounts and login processes will be based on requirements defined by WP7 "trustworthiness". As the portal relies on SFA mechanisms, a registry database will be needed. MySlice is able to handle several authentication sources. Partners will have the choice between trusting the Fed4Fire Registry as an authority that will have the right to register users for the whole federation or will have to peer their own registry to federate it (WP2 Architecture [1]). Note that the actual deployment of the Fed4FIRE registry is a task within WP7,

where the current evolution is to use a registry-only installation of SFA Wrap for this purpose. When registering new users under the authority of the Fed4FIRE Registry, then the portal provides a webpage where the user can fill in some basic personal information such as username, password, affiliation, and E-mail address. Once submitted by the user, the portal will be able to push this information into the Fed4FIRE Registry. Note that in the first cycle this registration page will not yet support advances features such as automatic verification of given personal data, the possibility to request additional personal information if this is needed by a specific testbed that the user wants to access, etc. However, these advanced features are considered to be important, and will be tackled in cycles 2 and 3.

Authentication

The portal will handle one login process according to WP7 requirements. The technology that will be adopted cannot be defined at the moment, since WP7 will only provide its requirements in D7.1, which has the same submission deadline as this deliverable D5.1. However, based on the ongoing discussions currently being held in WP7, some technologies seem to emerge as obvious candidates for this login process. These are to use OpenID or Mozilla Persona, which both are existing single-sign-on solutions, or to use the SFA certificate of the experimenter to gain access to the portal. In this last case, users should import that valid certificate, signed by a CA belonging to the federation, in their browser. Once this has been done, the user should be automatically authenticated when surfing to the portal, without needing to input a username and password. Of course, this also requires that the portal is aware of the root certificates of the different Fed4FIRE testbeds. In practice, this means that the portal has to be able to retrieve these from the Certificate Directory. However, as previously mentioned, a definitive technological choice and corresponding specification can only be made once WP7 has finalized its requirements and recommendations.

Next to the issue of portal login, authentication is also related to the usage of X.509 certificates in SFA. In short, Fed4FIRE experimenters must first of all have a valid public/private keypair. Based on these keys, they have to create a valid self-signed X.509 certificate. This certificate then has to be signed by the preferred identity provider. This provider has to be part of the Fed4FIRE. In practice most testbeds will operate their own identity provider. Besides, a central identity provider will be deployed within Fed4FIRE: the Fed4FIRE Registry. Once the certificate is signed, the experimenter can use this certificate for authentication when performing SFA API calls within the Fed4FIRE federation using a locally running user tool that is aware of the certificate. However, when the portal will act as a SFA user tool on behalf of the experimenter, then the user must delegate its credentials to the portal. In practice, this requires a small change to the X.509 certificate of the user, in order to enable delegation mode on the credential. Since user-friendliness is a key element for Fed4FIRE, this process should be hidden for novel experimenters as much as possible. Therefore it is needed that the portal knows how to automatically create the public/private keypair and the self-signed Fed4FIRE certificate. It should be able to push this self-signed certificate to the Fed4FIRE registry, and retrieve the version that is signed by the Fed4FIRE registry. Finally, the portal should be able to adapt this signed certificate to allow delegation. Based on current discussions being held in WP7, several technical routes can be followed when implementing this functionality. One is to implement this all in a Java applet which is started from

the portal but runs locally on the computer of the experimenter. This is more secure, since the private key never leaves the experimenter's computer. However, it is unclear if this approach is feasible in terms of efforts, since none of the consortium members have experience which such an approach. An alternative would be to create the keys and the certificate on the server side of the portal. This is just as convenient for the experimenter, but considered to be less secure since the private key will have been present on this server. However, for this approach several existing implementations are known, resulting in a higher confidence regarding the feasibility of this approach in terms of needed efforts. A final implementation approach will be defined once some more technical experience was gained with the different possibilities. This will be done in close cooperation with WP7.

Authorization & Access policies

Users that are registered (and manually approved on the Fed4FIRE registry) or already have an account on one testbed of the federation will be allowed to access the portal. They will also automatically gain the right to access Fed4FIRE testbeds which agreed to immediately provide access to users. However, some testbeds might have defined some specific authorization rules that might need more personal information about the experimenter (e.g. role within the organization: master student, PhD student, post-doc, professor, etc.) In that case the portal will request the experimenter to provide this additional information. In development cycle 1 of Fed4FIRE, only manual approval of new registrations on the Fed4FIRE registry will be supported. This is a responsibility of WP7. The functionality on the portal that is needed to support the rules-based authorisation is planned for cycles 2 and 3.

Testbed resources

The portal will present resources provided by the facilities in a user-friendly way and expose the diversity of testbed's resources through different plugins (tables, map, testbed map, detailed information about resources). The portal will allow users to browse, filter, add or remove resources in a slice and to create and update a slice across different testbeds of Fed4FIRE's federation. It will also provide the functionality to reserve resources, as specified in section 3.5. In order to include the resources of all Fed4FIRE's testbeds, these testbeds will be added to the portal and thoroughly tested in a sequential manner. Since this addition will always rely on the same SFA mechanisms, this is considered to be feasible in terms of needed efforts.

MySlice's Application Programming Interface

On its northbound interface, MySlice exposes itself to other tools through the MySlice API. Appendix A: Development status of MySlice API describes the development status of MySlice's API, API calls, actions and data formats.

3.2.6 Requirements for testbeds to interwork with the Portal

Testbeds aiming to integrate with the portal should comply with the following requirements:

- They need to expose their testbed through SFA (section 3.6), taking into account the specifications imposed by the Testbed directory (section 3.3), the Tools directory (section 3.4) and the Reservation broker (section 3.5).
- They should contribute to plug-ins development to expose specific features of their testbed (see MySlice Plugin Developer Website [22])

3.3 Testbed directory

3.3.1 General description

Fed4FIRE defines the testbed directory as follows: "A testbed directory which is readable by humans and by computers to have an overview of all testbeds in the federation", as stated in D2.1 (First federation architecture) [1]. Therefore we do not interpret the testbed directory here as a registry service where all resources accessible from the federated facilities are listed. In such a registry service, information on the capabilities of a particular resource would be included, as well as information on its requirements, e.g. in terms of interconnectivity or dependencies.

To be in line with the architecture that Fed4FIRE is pursuing, the intention is to make all critical services as distributed as possible. This means that resource discovery is performed by enabling direct interaction between the user tools and the different testbeds belonging to the federation. However, central components can be provided by the federation for the convenience of the experimenter.

To be more concrete, when an experimenter wants to discover resources, his/her user tool of choice will directly contact all known testbeds within the federation. Each testbed will list its own resources, including the capabilities of the specific resources, and the information on its requirements. The user tool of the experimenter will then combine all received lists in a single view. Now the question is: how will the user tool know whom to contact? One possibility is that the experimenter manually configures the IP addresses of the aggregate manager of all testbeds of his/her interest. This is a very reliable and sustainable solution, since this approach will remain functional even if the central federation service would be out of service and because, as explained in section 3.3.5 a scheduled task can automatically provide this information. As long as experimenters collect or exchange appropriate IP addresses, they will be able to continue using the different testbeds in the federation. However, if these lists of IP addresses would be made available in a central location, this makes it more convenient for the experimenter to make sure that all testbeds of the federation will be included in his/her discovery queries.

This is exactly the functionality that we expect the Testbed Directory to provide. It is a central service that provides the pointers to the different testbeds belonging to the Fed4FIRE federation. So in case of the machine readable flavour, it can be considered as a central service that exposes a list of IP addresses corresponding with the aggregate managers of the different Fed4FIRE testbeds. Very simply put: the machine readable flavour is a yellow pages of Fed4FIRE testbeds. In the human readable version, it provides textual high-level descriptions of these testbeds, optionally a picture, and a URL pointing to a particular testbed homepage where more detailed information is provided.

3.3.2 Evaluation of possible approaches for implementation

Different approaches for the implementation of the testbed directory have been considered. Both a clean slate design and the reuse of several existing software implementations and specifications have been taken into account.

The **first option** is to implement both the machine and the human readable version of the testbed directory from scratch as a central component. The main advantage of this approach is the guarantee that all requirements will be covered by such a clean-slate implementation. The disadvantage is that it is not the most efficient approach in terms of effort. It requires to develop the machine readable listing and webpage-based human readable version from the ground up, while it is not inconceivable that existing solutions could be suitable as an implementation basis. The fact that this option considers the testbed directory as a central component also leads to an operational overhead, since this requires a central collection and processing of all required inputs.

The **second option** that was explored is to implement the machine readable version from scratch, while linking to the training material on the project website for the human readable variety. Such an approach covers all requirements, and leverages the efforts invested in the definition and publication of the training material. On the short term this seems like a viable solution. However, on the long term it becomes clear that this is not a sustainable solution: after the end of the Fed4FIRE project, it cannot be guaranteed that the training material will be further updated. So, if the federation would continue to exist as a legal body and operational federation after the end of the project (which is one of the goals of the project) then additional efforts would have to be invested at the federation level in order to have an up to date testbed directory.

The **third option** is to start from the existing listing capabilities available in the MySlice API for the machine readable version. For the human readable version, the approach would be to include the corresponding information in the testbed-level part of the Rspec, and to create a MySlice plugin which extracts this information from the testbed Rspecs, and compiles an overview on a single webpage. This approach covers all imposed requirements, is more efficient in terms of effort than the previous options, and it is sustainable on the long term. The downside is that the extended Rspecs will become longer since it will contain all testbed directory information. This level of redundancy is however not needed nor desirable since these Rspecs are communicated from and towards the testbeds at a high rate.

The **fourth considered option** is to start from the existing listing capabilities available in the MySlice API for the machine readable version. For the human readable version, each testbed exposes the testbed descriptions (text, image, URL to more detailed information) on their SFA interface through the `GetVersion` SFA call. This call is normally used by SFA clients to query static configuration information about this aggregate manager implementation, such as API and RSpec versions supported. It can be considered as the SFA API call that provides testbed-level information. Therefore from all SFA API calls listed in the GENI Aggregate Manager API Version 3 specification [25], `GetVersion` is considered as the most appropriate place to expose the testbed descriptions that will be used by the testbed directory. Given that this information would be returned by each testbed as part of the `GetVersion` API call, Fed4FIRE could then develop a

MySlice plugin that extracts this information from the retrieved results, and compiles an overview on a single webpage. It inherits all advantages of the third option, while not extending the Rspecs with redundant information. The downside is that this approach requires the extension of the XML-RPC struct that is returned by the `GetVersion` SFA call. This means that there is a small deviation from the SFA API. However, this is not an issue; since the GENI API [26] states the following "Implementations can add additional members to the return struct as desired. The prefix geni_ is reserved for members that are part of this API specification. Implementations should choose an appropriate prefix to avoid conflicts. Aggregates should document any additional return values."

The fifth option taken into account is to start from the existing listing capabilities available in the MySlice API for the machine readable version. For the human readable version, each testbed exposes the testbed descriptions (text, image, URL to more detailed information) on their SFA interface through a new SFA API call that is specifically intended for the retrieval of a description of the testbed. This approach inherits all advantages of the fourth option, while not resulting in a deviation of the already defined SFA API calls. However, since new SFA methods will be required for integration in the Fed4FIRE federation, this approach loses the advantage of SFA compatibility. Therefore the needed actions should be taken to have these new API calls included in an official new release of the GENI Aggregate Manager API. This seems unfeasible within the stringent time constraints of cycle 1.

Therefore it is chosen to pursue solution 4 (extended struct returned by the SFA GetVersion method) in cycle 1, and to already start preparations to achieve solution 5 (introduce new specific methods to the SFA API) in cycle 2.

This analysis is summarized in Table 6:

Approach	Advantages	Disadvantages	Selected as final approach
Implement both from scratch as a central component	• Guaranteed to cover the requirements	• Not the most efficient way terms of effort • Overhead in central collection and processing of all information	
Implement machine version from scratch, link to the training material on the project website for human	• Guaranteed to cover the requirements • Leverage of the efforts invested in the definition and publication of the	• Not sustainable: additional efforts would have to be invested at the federation level once	

readable	training material	Fed4FIRe project is over	
Machine readable uses the MySlice API, human readable info in the Rspec	• Guaranteed to cover the requirements • Best reuse of previous implementation efforts • Sustainable	• Longer Rspec, which is communicated very often	
Machine readable uses the MySlice API, human readable info in the struct returned by `GetVersion`	• Guaranteed to cover the requirements • Best reuse of previous implementation efforts • Sustainable • No additions to Rspec	• Extended return value for the GetVersion call,hence alteration of SFA API	X
Machine readable uses the MySlice API, human readable info in new specific SFA calls	• Guaranteed to cover the requirements • Best reuse of previous implementation efforts • Sustainable • No additions to Rspec • No change to the existing API call implemenations	• Not longer fully compatible with SFA	

Table 6: Comparison of potential approaches for implementing the Testbed Directory

3.3.3 Description of selected tools

For the machine-readable version of the Testbed Directory, we will reuse the existing listing capabilities available in the MySlice API. Any tool or user will be able to send a Query to MySlice API which will list the platforms of the Fed4FIRE federation from its database and return the answer. The corresponding data has to be manually included once by the operator of the testbed directory. This operator can be the same as that of the portal, since it uses the same MySlice technology at its base. However, it is important to stress that this machine readable part of the Testbed Directory is considered to be a distinct part of the Fed4FIRE architecture, and not a part of the portal. Therefore it could also be operated by another party, which in this case would have to deploy a specific MySlice instance intended for the implementation of the Testbed Directory. But despite of the choice of operator, the testbeds themselves will always be responsible for the provisioning of this information to the testbed directory operator. Since this only has to be done once per testbed, no automation of this process is put in place.

For the human readable version, each testbed is expected to include the required information in the testbed-level part of its SFA driver. The human readable Testbed Directory will then be a MySlice plugin on the Fed4FIRE portal that contacts all testbeds listed in the machine-readable

version, retrieves the corresponding information, and displays it on a single webpage. To improve performance, this information will be cached by MySlice, and updated according to a regular update interval. This way, all testbeds do not have to be queried every time the Testbed Directory webpage is accessed. This regular update of the MySlice platform table will be performed by a periodic SFA `GetVersion` call to all Fed4FIRE testbeds.
Note that the option of extending the SFA API with a dedicated API call intended to retrieve the testbed description was identified in the previous section as the most desirable solution in the long run. But to maintain compatibility with SFA, it is needed that this extension is officially included in a new version of SFA. It will be investigated how the needed contacts can be established to pursue this option in cycle 2 and 3. For cycle 1 this approach is considered not feasible within the given time constraints.

3.3.4 Required additional implementations

Each partner should be SFA compliant and respond to the `GetVersion` API call with a description of the testbed. The XML-RPC struct returned by this call should be extended to include the needed information. Therefore the GetVersion method must be overridden in the SFA Driver of the Testbed.
Each partner will also have to provide minimal information to insert in the platform table of the MySlice database: name of the testbed and URL of its Aggregate Manager.
The MySlice database has to be extended to contain the testbed directory information. Besides, a MySlice update script will be needed to retrieve the descriptions from the testbeds on a regular basis and stores them in the database. Finally, a MySlice plugin has to be written that takes the data from the database and displays this information about the Fed4FIRE testbeds on a single webpage. The actual look and feel of this webpage will be specified during the development process, based on initial feedback by possible users on some initial mock-ups of the webpage.

3.3.5 Specifications

Application Programming Interface
The MySlice API can be queried through this python call which will return the needed information to construct the human-readable listing of the testbeds:

```
Query(action='get', fact_table='platform', filters=[], params=None,
fields=['platform', 'platform_description','platform_url_homepage',
'platform_url_picture'])

Similar, the query to retrieve the information related to the
machine readable directory looks as follows:

Query(action='get', fact_table='platform', filters=[], params=None,
fields=['platform', 'platform_url'])
```

The same calls can be triggered through an XMLRPC call:

```
# Connection to XMLRPC server
import xmlrpclib
srv = xmlrpclib.ServerProxy("http://hostname:7080/",
allow_none=True)

# Authentication token
auth = {"AuthMethod": "password", "Username": "mylogin",
"AuthString": "mypassword"}

srv.Get(auth, "platform", [], {}, ["platform",
"platform_description", "platform_url_homepage",
"platform_url_picture"])

srv.Get(auth, "platform", [], {}, ["platform", "platform_url"])
```

The description of the testbed stored in the MySlice database will rely on the GetVersion SFA call [27]. This API call remains exactly the same as defined by the GENI Aggregate Manager API Version 3, the only difference is that the returned XML-RPC struct is extended as described in the next section.

GetVersion
Syntax: struct GetVersion([optional: struct options]) **Functionality:** Get static version and configuration information about this aggregate. Return includes: • The version of the GENI Aggregate Manager API supported by this aggregate manager instance. • URLs for other versions of this API supported by this aggregate • The RSpec formats accepted at this aggregate • Other information about the configuration of this aggregate. • Fed4FIRE extension: short description of the testbed, URLs to picture and homepage, list of tools that are officially endorsed by the testbed. **Parameters:** options: Optional **Returns:** Originally a struct where the value member is Version Information, in the Fed4FIRE context this same struct is extended with information needed by the testbed directory and the tools directory. Such extensions are allowed in the context of SFA.

Data Formats

The MySlice internal Database will store the testbeds list. The information related to the machine readable variety will be entered manually by the database manager. This will be the name of the testbed, and the corresponding IP address of its aggregate manager.

Testbeds descriptions regarding the human readable version of the testbed directory will be cached in the same database, as this information does not change often. It will be updated periodically through a scheduled task. This automatic task will be implemented in cycle 1, and will periodically perform a SFA GetVersion call to all testbeds that were manually entered in the MySlice database in the context of the machine readable version of the directory. Based on the retrieved information, the database will be updated accordingly. The corresponding table in the database will contain the following fields:

```
0|platform_id|INTEGER|1||1
1|platform|VARCHAR|0||0
2|platform_longname|VARCHAR|0||0
3|platform_description|VARCHAR|0||0
4|platform_url|VARCHAR|0||0
5|deleted|BOOLEAN|0||0
6|disabled|BOOLEAN|0||0
7|status|VARCHAR|0||0
8|status_updated|INTEGER|0||0
9|platform_has_agents|BOOLEAN|0||0
10|first|INTEGER|0||0
11|last|INTEGER|0||0
12|gateway_type|VARCHAR|0||0
13|gateway_conf|VARCHAR|0||0
14|auth_type|VARCHAR(9)|0||0
15|config|VARCHAR|0||0
16|platform_url_homepage|VARCHAR|0||0
17|platform_url_picture|VARCHAR|0||0
```

Note that several of these fields are not directly needed by the Testbed Directory, but were already present in this MySlice data model to support other MySlice functionalities.

Extended XML-RPC struct returned by GetVersion
In the GENI AM API v3, it is defined that the GetVersion SFA API call returns a XML-RPC struct that describes which version of the Aggregate Manager API is running locally, the RSpec schemas supported, and the URLs where other versions of the AM API are running. In Fed4FIRE, this XML-RPC struct is extended with information related to both the testbed and the tool directories. In order to be as clear as possible in this specifications document, we include both extensions in this section regarding the testbed directory. In the section about the tool directory, we will refer to the extended definition given below. The extension defined in this document is related to the member of the returned XML-RPC struct

that is called `value`. The members `geni_api`, `code` and `output` are not affected by the Fed4FIRE extension.

The `value` member by itself is again an XML-RPC struct. It contains the following members:

- `geni_api`: an integer indicating the revision of the Aggregate Manager API that an aggregate supports. In this deliverable we are extending version 3 of the API
- `geni_api_versions`: an XML-RPC struct indicating the versions of the Aggregate Manager API supported at this aggregate, and the URLs at which those API versions can be contacted. This element is required, and shall include at least 1 entry indicating the local aggregate manager URL and the version of the API supported at that URL.
- `geni_request_rspec_versions`: an array of data structures indicating the RSpec types accepted by this AM in a request
- `geni_ad_rspec_versions`: an array of data structures indicating what types of RSpec advertisements may be produced by this AM in ListResources.
- `Geni_credential_types`: not documented on the GENI AM API v3 wiki page, but included in its sample output. Therefore it is mentioned here. In the example, this is an array of data structures that indicate the credential types accepted by this AM.
- `geni_single_allocation`: not documented on the GENI AM API v3 wiki page, but included in its sample output. Therefore it is mentioned here. In the example, this is an integer that indicates if the AM can operate on individual slivers.
- `geni_allocate`: not documented on the GENI AM API v3 wiki page, but included in its sample output. Therefore it is mentioned here. In the example, this is a String that indicates if the AM can do multiple Allocates.

To support the Fed4FIRE testbed and tools directories, the `value` XML-RPC struct is extended with the following members:

- `f4f_describe_testbed`: a String containing a human-readable description of the testbed. The intention of this field is to provide rather high-level introductory information to the testbed.
- `f4f_testbed_homepage`: a String containing the absolute URL to the testbed homepage or any other webpage that provides more detailed information about the testbed than what will be included in the member `Fed4FIRE_describe_testbed`.
- `f4f_testbed_picture`: a String containing the absolute URL to a picture of the testbed that should be presented in the overview of all Fed4FIRE testbeds as presented by the testbed directory.
- `f4f_endorsed_tools`: an array of data structures indicating the tools that are officially endorsed by the testbed. This data structure contains the following members
    - `tool_name`: a String that contains the name of the tool.
    - `tool_logo`: a String containing the absolute URL to the logo of the tool.
    - `tool_homepage`: a String containing the absolute URL to the tool homepage or any other webpage that provides more detailed information about the tool.
    - `Tool_version`: a String indicating the latest version of the tool that is officially endorsed.

Sample output (Fed4FIRE extension annotated in green)

```
{
  geni_api = 3 # This is AM API v3
  code = {
       geni_code = 0 # Success
       # am_type and am_code are optional. Leaving them out.
       }
  value =
       {
       geni_api = 3 # Match above
        geni_api_versions = {
              '3' = <This server's AM API absolute URL>
              '2' = <Prior API version still supported at a slightly different URL
- optional but included here>
       }
        geni_request_rspec_versions = [{
             type = "GENI" # case insensitive
             version = "3" # case insensitive
             schema    =    "http://www.geni.net/resources/rspec/3/request.xsd"    #
required but may be empty
             namespace = "http://www.geni.net/resources/rspec/3" # required but may
be empty
             extensions   =   ["http://hpn.east.isi.edu/rspec/ext/stitch/0.1/stitch-
schema.xsd", <other URLs here>] # required but may be empty
       }]
        geni_ad_rspec_versions = [{
             type = "GENI" # case insensitive
             version = "3" # case insensitive
             schema = "http://www.geni.net/resources/rspec/3/ad.xsd" # required but
may be empty
             namespace = "http://www.geni.net/resources/rspec/3" # required but may
be empty
             extensions   =   ["http://hpn.east.isi.edu/rspec/ext/stitch/0.1/stitch-
schema.xsd", <other URLs here>] # required but may be empty
       }]
        geni_credential_types = [{ # This AM accepts only SFA style credentials for
API v3
             geni_type = "geni_sfa" # case insensitive
             geni_version = "3" # case insensitive
       }]
       geni_single_allocation = 0 # false - can operate on individual slivers. This
is the default, so could legally be omitted here.
       geni_allocate = "geni_many" # Can do multiple Allocates. This is not the
default value, so is required here.
        f4f_describe_testbed = "This is a specific description of a particular
testbed. In this text it is briefly explained what it is about, on a high level.
This is exactly the text that will be displayed by the testbed directory."
        f4f_testbed_homepage = "http://www.afederatedtestbed.eu"
        f4f_testbed_picture = "http://www.afederatedtestbed.eu/overview.jpg"
        f4f_endorsed_tools = [{
             tool_name = "Tool X"
             tool_logo = "http://www.firetoolx.eu/logo.jpg”
             tool_homepage = "http://www.firetoolx.eu"
             tool_version = "3.2.2"
```

```
 },{
              tool_name = "Tool Y"
              tool_logo = "http://www.firetooly.eu/logo.jpg"
              tool_homepage = "http://www.firetooly.eu"
             tool_version = "2.1.2"
 }]
       }
  output = <None>
}
```

3.3.6 Requirements for testbeds to adopt the specified testbed directory solution

Testbeds aiming to support this functionality should comply with the following requirements:

- They should expose their testbed through SFA
- They should extend the XML-struct returned by the `GetVersion` call as defined in this deliverable. More concrete it should contain a small description, and links to an online image and the testbed homepage.
- They should make sure that their AM runs on a static IP address, and they have to provide this address to the operator of the Testbed Directory

3.4 Tool directory

3.4.1 General description

Fed4FIRE defines the tool directory as follows: "A tool directory which gives an overview of available tools for the experimenter". Taking this definition as a starting point, the first question to answer is: what is the scope of this directory? Is it limited to an enumeration of the tools especially developed to enable federation within Fed4FIRE? Examples of such tools could be the portal, the reservation engine, the testbed directory and the certificate directory. Another approach could be to extend the scope of the tool directory towards all FIRE tools that could be applied usefully on any of the Fed4FIRE federated facilities. This could be any tool, both those that were adopted officially by the federated testbeds and those that were unknown to the testbed owners but identified by experimenters and/or tool providers as working on the federated testbeds.

In consultation with the entire project consortium, the scope of the tool directory was defined to be of the second category. The tools directory should provide pointers to FIRE tools of all kinds, both to the officially endorsed ones and to those tools that emerged naturally from the FIRE community. However, some mechanism should be in place in order to enable testbed providers to indicate the tools that they officially endorse on their testbed.

3.4.2 Evaluation of possible approaches for implementation

Different approaches for the implementation of the tools directory have been considered. The **first option** is to merely provide a link to the training material. Such an approach is of course very efficient in terms of needed effort. However, it does not reach the intended scope, since the training material is limited to tools especially developed to enable federation within Fed4FIRE. Besides, this approach results in a management overhead for the federation, since all tools information has to be maintained in a single central component. It is also not sustainable: after the end of the Fed4FIRE project, it cannot be guaranteed that the training material will be further updated. So if the federation would continue to exist as a legal body and operational federation, which is the objective of Fed4FIRE, additional efforts would be needed at the federation level in order to have an up to date tools directory.

The **second considered option** is to adopt the same approach as for the testbed directory: all tool builders expose the information regarding the tools through an SFA interface. The benefit of this approach is the fact that all tools within the scope can be covered, and that the information is maintained by the tool owners. Hence there is no central management overhead, and this solution is considered sustainable on the long term. However, this approach seems rather unpractical. In this case, it would be required that each tool provider sets up an AM to provide information about the tool. However, an AM is intended for the management of a testbed, not the dissemination of information. Another disadvantage is the fact that experimenters cannot provide information. Similar to the approach taken for the testbed directory, a minor deviation of the SFA API (extension of the struct returned by the `GetVersion` API call) can also here be considered as a drawback.

The **third possible approach** is to gather all tools-related information on a public Wiki page on the Portal. As a result, the Tool Directory would become a part of the Portal from an architectural point of view. This approach inherits the same advantages as the previous option: all tools within the scope can be covered, no central management overhead exists and it is sustainable on the long term. Moreover, this solution is straightforward to implement, and allows experimenters to contribute. The only disadvantage is the fact that since anyone can edit, it is hard to distinguish between official Fed4FIRE information (tools endorsed by each testbed) and information provided by the FIRE community.

The **last considered approach** is a combination of the previous two: the testbeds expose officially endorsed tools through their SFA interface, while a public Wiki page is foreseen on the public section of the Fed4FIRE Portal for gathering of information about all FIRE tools. This way the advantages of solution three can be kept, while having a clear distinction between official Fed4FIRE and FIRE community information. The only disadvantage that remains is the minor deviation of the SFA API, more specific in the return value of the `GetVersion` API call. However, as also mentioned in case of the testbed directory, this is not an obstacle since GENI explicitly states that returned XML-RPC structs can freely be extended as long as the extension is well annotated and documented.

Therefore this last approach was adopted for cycle 1. Similar to the testbed directory, cycles 2 and 3 will also target the official extension of the SFA API with an API call especially intended to list the tools endorsed by a testbed.

Approach	Advantages	Disadvantages	Selected as final approach
Link to training material	• Optimal reuse of manpower	• Limited to tools especially developed within Fed4FIRE • Overhead of central data maintenance • Not sustainable.	
Adopt testbed directory approach: have all tool builders expose info through SFA	• Can cover the entire scope • Information is managed by the tool builders: no central overhead and sustainable on the long term.	• Unpractical. This would require that each tool provider sets up an AM to provide information about the tool. • Experimenters cannot provide information. • Extended return value for the GetVersion	

		call,hence alteration of SFA API	
Gather all tools-related information on a public Wiki page on the Portal	• Can cover the entire scope • Information is managed by the tool builders: no central overhead and sustainable on the long term. • Straightforward to implement • Experimenters can contribute	• Hard to distinguish between official Fed4FIRE information and information provided by the FIRE community.	
The testbeds expose officially endorsed tools through their SFA interface, public Wiki page on the Portal for gathering of information about all tools	• Can cover the entire scope • Information is managed by the tool builders: no central overhead and sustainable on the long term. • Straightforward to implement • Experimenters can contribute • Clear distinction between official Fed4FIRE and FIRE community information.	• Extended return value for the GetVersion call,hence alteration of SFA API	X

Table 7: Comparison of potential approaches for implementing the Tool Directory

3.4.3 Description of selected tools

Regarding the exposure by each testbed of its endorsed tools through the SFA API call `GetVersion`, it is sufficient to refer to the section regarding the testbed directory. It was already explained there that this method is the most suitable place in the SFA API to provide information about the testbed itself. For the tools directory, the returned data structure is extended to include the information about the tools endorsed by the testbed. The MySlice plugin that provides the testbed directory compilation will be able to annotate for each testbed which tools are officially endorsed. Hence from the experimenter point of view, this part of the tools directory is in fact displayed as a piece of the testbed directory web page.

The public Wiki page on the other hand will be a separate section on the portal, clearly identifiable as a tools directory that gathers inputs from the entire FIRE community. Wiki is considered as a sound technical basis for such a public forum, since it has a proven track record in many other projects regarding information gathering and collaborative document writing

(Wikipedia being one of the most famous examples). In fact, Ward Cunningham, the developer of the first wiki software, described the essence of the Wiki concept [28] as follows:

- A wiki invites all users to edit any page or to create new pages within the wiki Web site, using only a plain-vanilla Web browser without any extra add-ons.
- Wiki promotes meaningful topic associations between different pages by making page link creation almost intuitively easy and showing whether an intended target page exists or not.
- A wiki is not a carefully crafted site for casual visitors. Instead, it seeks to involve the visitor in an ongoing process of creation and collaboration that constantly changes the Web site landscape.

3.4.4 Required additional implementations

Regarding the endorsed tools, the to-be-developed MySlice plugin for the testbed directory should include the functionality needed by this part of the tools directory. More concrete: for every testbed that is displayed, a list of endorsed tools should be extracted from the data structure returned by the `GetVersion` call, and annotated on the webpage. The available information regarding each tool is its name, URL to its homepage and URL to its logo. The MySlice plugin should annotate this information to each testbed displayed in the testbed directory in a suitable manner. Each testbed of course should make sure that when overriding their GetVersion return method as demanded by the testbed directory, that they also include this information regarding endorsed tools.

Regarding the Wiki page, no real implementation work is needed. The Fed4FIRE portal will be based on MySlice, which on its turn is based on the Joomla content management system (CMS) [20]. This is a very popular CMS, many extensions for it can be downloaded as open source software from the Internet. Such extensions that enable a Wiki system within the Joomla framework already exist. One example is JWiki [31]. And even if the integration of a wiki inside the MySlice-based Portal would prove to be harder than anticipated, an acceptable backup plan would be to setup a dedicated wiki for the tools directory (e.g. a MediaWiki installation), and link to it from the Portal.

The most important part of the Wiki installation will be the initial structure of the content. Here any tool maker will be able to post some information regarding the functionality of the tool, on which testbeds it has been reported working, link to the homepage, etc. It is not inconceivable that without a good structure, the amount of content will quickly become too large to result in a usable tool directory. Therefore the base content structure of this Wiki-based tools directory is crucial. It should make a clear distinction between sections intended for novel experimenters, for experienced experimenters, and for testbed owners/developers. Keeping an eye on this structure during the course of the Fed4FIRE project (and beyond) can be considered as one of the biggest efforts needed for the Wiki portion of the tools directory.

3.4.5 Specifications

The listing of endorsed tools through each testbed's SFA interface requires an extension of the data structure returned by the SFA API call `GetVersion`. However, the approach adopted in the context of the testbed directory also extends this specific data structure. To avoid any possible confusion by defining the extension of this data structure in two different parts of this deliverable D5.1, it was chosen to gather all extensions to this data structure in one place: the section regarding the testbed directory. Hence the reader is referred to that section for the specification of the data structure extensions related to the listing of the endorsed tools. Note that this is the only change needed to the SFA API to support the listing of endorsed tools.

Regarding the usage of a public Wiki page on the Fed4FIRE portal, no specifications can be defined, since no API is provided or needed by this component. Regarding editing, all pages will be unprotected, meaning that anyone can write to them. The editing policy of Wikipedia is adopted [108]. In terms of the structure of the wiki, a first draft is given below. Note that this will evolve over time as more and more content is being put on the wiki by the FIRE community members.

- Novel experimenters
  - Graphical resource discovery, reservation and provisioning tools
    - Fed4FIRE portal
    - FITeagle [55]
    - Flack [12]
- Experienced experimenters
  - Command line resource discovery, reservation and provisioning tools
    - SFI [29]
    - Omni [29]
  - Experiment control tools
    - OMF [63]
    - NEPI [8]
  - Monitoring and measuring tools
    - OML [64]
    - Zabbix [65]
    - CollectD [66]
- Testbed owners and developers
  - SFA support
    - SFA wrap [60]
    - AMsoil [62]
    - SFA test suite
  - Reservation engines
    - Fed4FIRE future reservation broker (see section 3.5)
  - MySlice plugin development [19]

3.4.6 Requirements for testbeds to adopt the specified testbed directory solution

Testbeds aiming to support this functionality should comply with the following requirements:

- They should expose their testbed through SFA
- They should extend the XML-struct returned by the `GetVersion` call as defined in section 3.3. More concrete it should contain a list of endorsed FIRE tools. For each tool, the following information is needed: name, and links to an online hosted logo and the tool homepage.
- If they are also tool builders, they should create an informational section about their tools on the Wiki part of the tool directory.

3.5 Future reservation broker

3.5.1 General description

The goal of this component is to provide the functionality of a reservation system for various reservation types of resources in Fed4FIRE facilities. The outcome is an overarching service that experimenters can utilize to reserve resources, which in turn will contact each individual facility and ask for a reservation. The taxonomy of the various types of reservation is provided in Table 8.

The distinction between the various types of reservation is made upon two different dimensions: (1) time and (2) guarantee of resources. With reference to the time-based dimension, we differentiate between instant, advance and elastic reservations, while for the guarantee of resources dimension we differentiate between hard and best-effort reservations. An intersection of the two dimensions is possible (e.g. instant reservation for resources provisioned on a best-effort basis). Considering the two dimensions identified for resources reservation, different service levels will be supported. For example, based on the time dimension, resources reservation service may be denoted as silver, gold, and premium to characterize its duration. Similarly the guarantee of resources dimension may be used to define different service levels. However service level differentiation will be looked into detail at a following cycle of the broker implementation. **Note that based on this distinction, the name "Future reservation broker" in fact is not entirely suitable, since this component will at least support both instant and advanced reservations. However, to be aligned with D2.2 (First federation architecture), we maintain the naming as defined there, since in this deliverable D5.1 we are actually defining the specifications of D2.2. However, in the next iteration of the architecture we will make sure that this component is renamed to "Reservation broker".**

	Type	Description
Guarantee of resources	**Hard Reservation**	Guaranteed resources availability (usually realized by exclusive access on the resource).
	Best-effort Reservation	Resources are provisioned on a best-effort basis, shared among allocated experimenters' requests (non-exclusive access on the resource).
Time	**Advanced Reservation**	Advanced reservation mechanisms of resources by the experimenters utilizing a calendar service.
	Instant Reservation	Resources are reserved from the time the experimenter's request arrives at the reservation system.
	Elastic Reservation	The time constraint is not hard, the reservation system may look for the 1st available times slot where all resources are

		available (within a pre-specified time interval and for a given duration).

Table 8: Taxonomy of reservation types

In the Fed4FIRE federation environment, experimenters will be able to reserve resources via (Figure 6):

1. **The resource brokering service** (Future Reservation Broker) that will be integrated into the central Fed4FIRE facility. The Future Reservation Broker will support all types of reservations, described in Table 7, by helping to match and optimize the timeframe and resources requirements, set by the experimenter over one/multiple testbeds.

2. **Appropriate experimenter tools**. Experimenters may use existing tools to either directly reserve resources that belong to a specific testbed (e.g. using testbed-specific experimenter's tool) or use appropriate tools that allow directly reserving resources from federated testbeds e.g. SFI [29].

Based on the time dimension, the testbeds that are part of theFed4FIRE federation environment are primarily classified (Table 8) to facilities that already make use of a scheduling service and facilities with an online system for immediate reservation of resources. Moreover on Table 8 we identify the testbed's functionality regarding the guarantee of resources dimension.

Facility	Instant Reservations	Advance Reservations	Elastic Reservations	Hard Reservation	Best-Effort Reservation
PlanetLab [33]	X	X(Reservable Nodes)	-	X(Reservable Nodes)	X
NITOS [34]	-	X	-	X	-
VIRTUAL WALL [35]	X	-	-	X	-
FuSeCo [37]	X	-	-	X	-
OFELIA [59]	X	-	-	-	X
w-iLab.t [36]	X	-	-	X	-
NETMODE [38]	-	X	-	X	-
NORBIT [39]	-	X	-	X	-

SmartSantander [41]	X	-	-	-	X
GRID5000 [40]	X	X	X	X	-
Bonfire [10]	X(Virtual Machines)	X(Physical Machines)	-	X	-
NIA/KOREA [42]	-	X	-	X	-

Table 9: Currently supported reservation types of Fed4FIRE's testbeds

Considering that both testbed types (testbed with/without a scheduling service) are SFA-enabled, the Broker interacts with the testbeds via the SFA interface. The aforementioned functionality is depicted in Figure 6. Specifically, in the first case, the scheduling service of the testbed acts as a slave broker for the Future Reservation Broker (master). In the latter case, the testbed must export a pre-selected set of testbed resources to the Future Reservation Broker, or deploy a local scheduling solution which can act as a slave broker for the Future Reservation Broker.

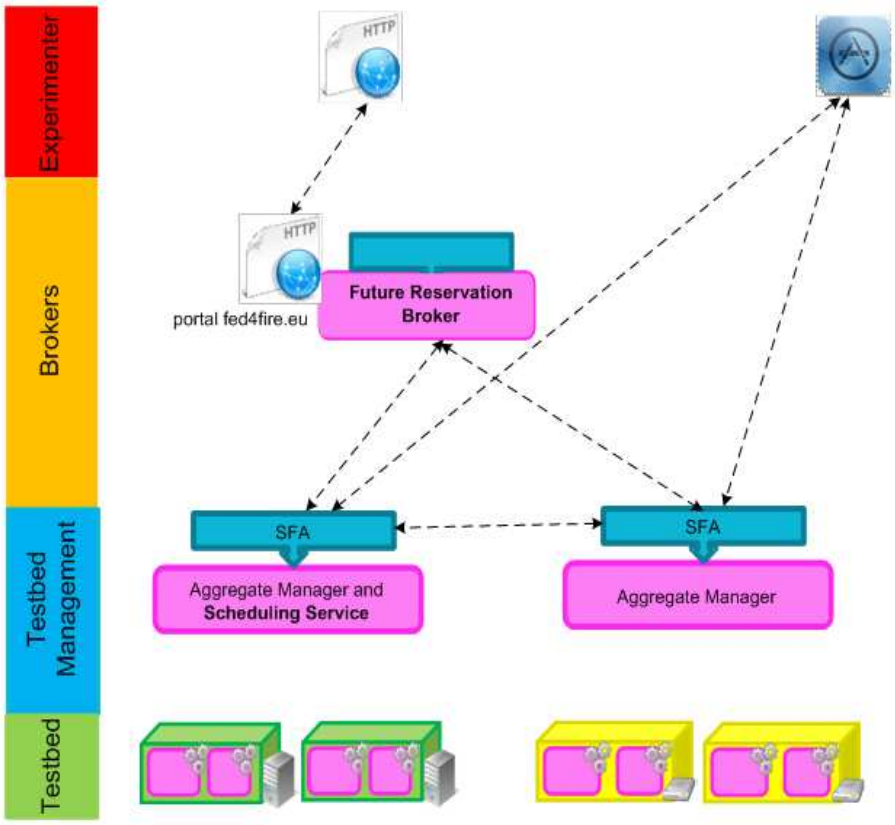

Figure 6: Fed4FIRE Reservation System

The main components that the Future Reservation Broker will interact with are the following, listed here in a top-down fashion, as depicted in Figure 7:

1) **Portal**: The authenticated user via the portal interacts with the Future Reservation Broker in order to discover the available resources from the federated testbeds. The user, at the end of cycle 1 will be able to make in advance reservation of resources utilizing the broker.

2) **Identity provider:** During the resource discovery phase, resources for each authenticated user are filtered according to rules set by the testbed providers on user access.

3) **Certificate Directory**: In order to retrieve the appropriate information from the identity provider, the Broker needs to establish a chain of trust with testbeds' identity providers.

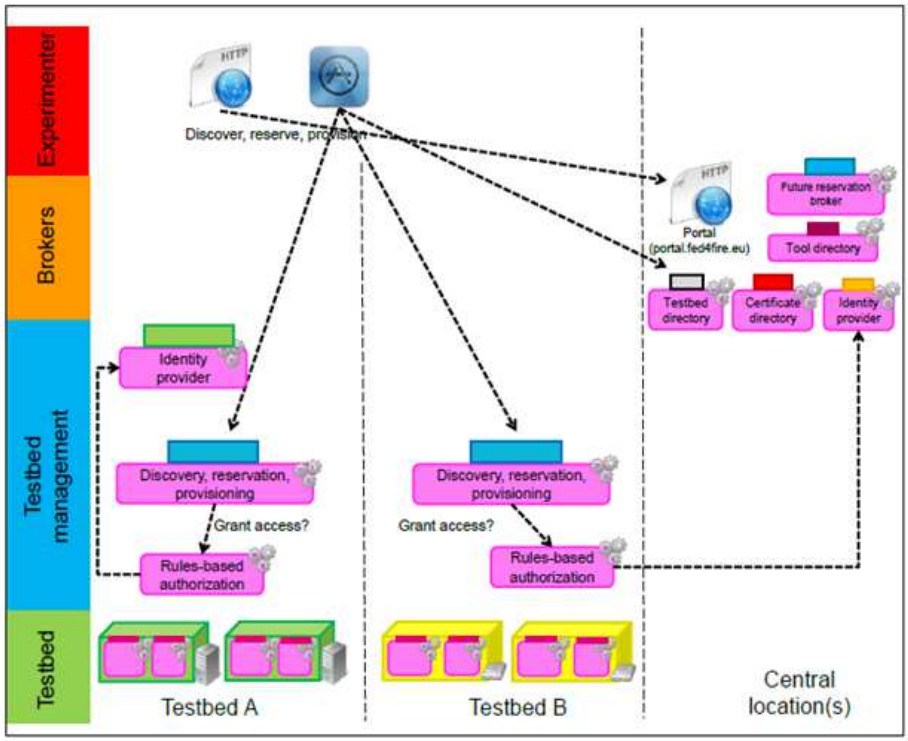

Figure 7: Future Reservation System interactions with Fed4FIRE central location components

3.5.2 Evaluation of possible approaches for implementation

Three different approaches were evaluated for the future reservation broker functionality. All approaches are based on the adaptation of existing tools/software. The three tools that were examined are the following:

NITOS/NICTA Broker: The NITOS Scheduler [43] has evolved along with OMF [63] and with the collaboration of NICTA to what is called a "Broker". This name was chosen to indicate the extra functionalities beyond scheduling of resources reservation. The purpose of the joint work with NICTA was to provide an inherent scheduling functionality to OMF and capabilities like an SFA API for easy federation of OMF testbeds.

The architecture includes components covering the following functionalities:

- Communication (Broker's main API)
- Authentication / Authorization
- Scheduling
- Liaison to the underlying resources

Even though the Broker has been designed in a way to work flawlessly with OMF, it doesn't retain any strict connections to it. The modular design, enables the Broker to be flexible enough in order to adapt in platforms that expose a different API than the OMF6 messaging protocol (SFA is one of the alternative options).

The NITOS broker can be also used in a hierarchical reservation system where a main Broker can be contacted directly from a client and reserve resources that have been assigned to the main Broker by another (slave) Broker.

GRID5000 Scheduler: At its core, Grid'5000 [40] uses OAR as a scheduler [44]. The implementation chosen is to run an independent OAR instance on each site, and to wrap OAR's REST API into a slightly higher level REST API specific to Grid'5000.

OAR is a batch scheduler and resource manager. As a scheduler, it supports many standard features: interactive jobs (get resources now), batch scheduling (run the following job (maybe sleep to give control to the experimenter) when resources fitting the description are available and advance reservation. Moreover, OAR supports best-effort jobs, where resources are only made available to a user as long as no standard usage is requested). As a resource manager, it will configure resources at the start of jobs to ensure exclusive access to the job owner, and clean-up at the end of the job to ensure a clean state for the next user. Also, it will detect problems with resources (when they are no longer reachable or exhibit incorrect behaviour) and adapt scheduling to take into account these suspect nodes. OAR has a simple notion of users: it will keep track of a login associated to each job, and configure a pair of SSH keys to be used by that user to gain exclusive access to nodes. It can also be instructed to use the user's standard public key (or any public keys chosen by the submitter). There is no support for signed assertions of x.509v3 certificates.

As Grid'5000 is running OAR on each site, some tools are provided to help experimenters use resources on more than one site. For each resource, the current status (free or used) and future reservations are made available to users over Grid'5000's REST API, so they can build their own broker. For example, OAR-GRID is a tool to reserve resources from different sites of Grid'5000. OAR-GRID's principle is very simple, on each wanted clusters a reservation is preceded (in OAR). If one of them does not succeed then all previous reservations are cancelled as well as the global operation. Moreover, a grid resources discovery tool (called disco) is also available. The tool enables resources discovery at a precise date (including now) for any amount of time.

Finally, a thin REST API for Grid'5000 has been developed over the native OAR REST API that provides notifications to inform users or services when a job starts. This notification system can send mails, XMPP messages or POST requests to user specified URLs.

NETMODE Scheduler: The service oriented wireless resource management framework, denoted as NETMODE scheduler [38], is deployed at the Network Management and Optimal Design Laboratory of the National Technical University of Athens, Greece. NETMODE Scheduler adopts spectrum slicing which enables the co-existence of multiple virtual topologies, with minimum interference via efficient spectrum utilization. Specifically an experimenter can only use nodes and wireless channels that have not been reserved by another user for the same time period.
The architecture includes components covering the following functionalities:

- Communication (NETMODE Scheduler Web Services API)
- Authentication / Authorization
- Scheduling

The NETMODE Scheduler utilizes the OMF resource provisioning framework. Moreover, a formal description of resources is used in order to create a system capable of selecting, reserving and configuring resources in an automated manner. Therefore a resource specification language (ProtoGENI V2 format RSpec) has been adopted and extended for wireless experimentation.

A summary of the advantages and disadvantages of the examined approaches is illustrated in Table 10:

Approach	Advantages	Disadvantages	Selected as final approach
NITOS/NICTA Broker/Scheduler	• Integrated with OMF (in progress) • Provides SFA support • Supports PlanetLab extended RSpecs • Provides hierarchical scheduling • Reasonable extra implementation effort needed • Open source (No issues with IPR)	• Partially supported RBAC	X
GRID5000 Scheduler	• RESTful API • Admission rules for policy	• Does not provide SFA functionality • Large implementation	

	• Mature technology	effort needed	
NETMODE Scheduler	• Web services API (SOAP) • Supports ProtoGENI v2 RSpecs • Open source (No issues with IPR)	• Does not provide SFA functionality • Large implementation effort needed	

Table 10: Comparison of potential approaches / eligible tools for realising Fed4FIRE's reservation system

As we can observe from Table 10, the NITOS Broker/scheduler is the most complete scheduling system in terms of the provided functionalities (SFA enabled, use of RSpecs, integrated with OMF, hierarchical scheduling) compared to the other two solutions, whereas it can be easily extended to cover the requirements for the future reservation system as posed in D2.1 [1].

3.5.3 Description of selected tools

In this section, the architectural design and supported functionalities of the NITOS Broker are presented in detail. As mentioned in the previous section, the NITOS Broker consists of several modules, each of them responsible for different functionalities. Figure 8 depicts the main components of the Broker and their interactions. The Broker entity is deployed in the testbed and is responsible for brokering testbed's resources. The different AMs that can be seen in Fig. 11 can be part of the same testbed/organisation (e.g. an AM for an indoor testbed and an AM for an outdoor testbed) or can belong to different testbeds/organizations. The latter refers to the hierarchical scheme of the Broker where a "master" Broker resides outside of a testbed and acts on behalf of the "slave" Broker which is deployed in organization's testbed. This kind of deployment is scheduled for the second phase of cycle 1 when the Authentication/Authorization scheme will be agreed among all testbeds such that A/A could take place in a central location.

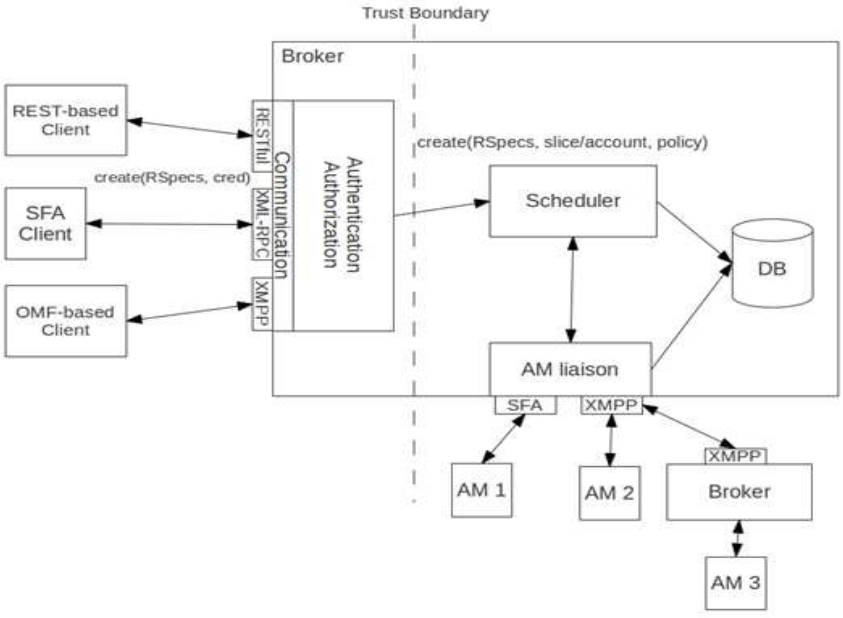

Figure 8: NITOS Broker Architecture

Communication

The Communication block concentrates all the available interfaces (APIs) of the Broker. This is the main entry point for the client queries with respect to resource discovery, reservation and provisioning. The Communication block is comprised of an XML-RPC interface for SFA clients, an XMPP interface based on the pubsub service of the OMF messaging system and a REST API for other uses if needed, described in the following in more detail:

XML-RPC

The XML-RPC [45] is the basic interface for the SFA clients and supports the methods implied by the latter. Right now the supported version is that of the ProtoGENI [78] implementation, while the PGv2 RSpecs [80] have been extended to support in advance reservation information. Ultimately it will support the SFA flavour that it will be decided for the Fed4FIRE project. In the following examples of advertisement (Table 11) and request (Table 12) RSpecs are provided. Manifest RSpecs at the time are implemented as advertisement RSpecs including the resources of the slice.

```
<?xml version="1.0"?>
<rspec xmlns="http://www.protogeni.net/resources/rspec/2"
xmlns:omf="http://schema.mytestbed.net/sfa/rspec/1"
type="advertisement" generated="2012-12-18T18:08:08+02:00"
expires="2012-12-18T18:18:08+02:00">
  <node id="aea0b9a5-e90e-5fd6-9224-847f0a1b37cb"
omf:href="/resources/aea0b9a5-e90e-5fd6-9224-847f0a1b37cb"
component_id="urn:publici
d:IDN+mytestbed.net+node+aea0b9a5-e90e-5fd6-9224-847f0a1b37cb"
component_manager_id="authority+am" component_name="n0">
      <available now="true"/>
  </node>
  <node id="70dce487-582e-5e56-a3ed-a1541ed73826"
omf:href="/resources/70dce487-582e-5e56-a3ed-a1541ed73826"
component_id="urn:publici
d:IDN+mytestbed.net+node+70dce487-582e-5e56-a3ed-a1541ed73826"
component_manager_id="authority+am" component_name="n1">
      <available now="true"/>
  </node>
  <node id="2564c0a3-dd23-551f-a25c-1af767f45a81"
omf:href="/resources/2564c0a3-dd23-551f-a25c-1af767f45a81"
component_id="urn:publici
d:IDN+mytestbed.net+node+2564c0a3-dd23-551f-a25c-1af767f45a81"
component_manager_id="authority+am" component_name="n2">
      <available now="true"/>
  </node>
</rspec>
```

Table 11: Advertisement RSpec – NITOS/NICTA Broker

In the following Table 12 an example of request RSpecs is provided.

```
<?xml version="1.0"?>
<rspec xmlns="http://www.protogeni.net/resources/rspec/2"
xmlns:omf="http://schema.mytestbed.net/sfa/rspec/1" type="request"
xmlns:olx="http://schema.nitlab.inf.uth.gr/sfa/rspec/1">
  <olx:lease lease_name="l1" olx:valid_from="2013-01-08T20:00:00Z"
olx:valid_until="2013-01-08T20:00:00Z"/>
  <node component_id="urn:publicid:IDN+openlab+node+node1"
component_name="node1" olx:lease_name="l1">
  </node>
</rspec>
```

Table 12: Request RSpec – NITOS/NICTA Broker

Furthermore, this interface can be also used for linking Brokers such that the "master" Broker will send requests through the XML-RPC interface to the "slave" Broker. More details for the hierarchical use of the Broker will be in a following section.

XMPP

The XMPP [79] interface is used by OMF-based clients (utilizing the OMF6 messaging protocol). Specifically the "create" and "inform" messages are used. Specifically, the user will ask the Broker to "create" some resources and the latter will "inform" about the success or not of the request. In the same fashion as in the XML-RPC case, PGv2 RSpecs are used for resource discovery, reservation and provisioning.

Furthermore, this interface can be also used for linking Brokers such that the "master" Broker will send requests through the XMPP interface to the "slave" Broker. More details for the hierarchical use of the Broker will be described in a following section.

RESTful

The design of the communication block allows having also a RESTful [81] interface. This allows for 3rd party clients can be supported at later stages.

Authentication / Authorization

The "A/A" component is the policy enforcement point of the testbed. Its main purpose is to check the user's incoming messages, depending on the communication API used, and set the authorization context. After the reception of a message the "A/A" parses the credentials and extracts the necessary information for policy enforcement.

The credentials in SFA consist of an owner and a target (X509 certificate based), expiry time and a list of zero or more privileges. The same concept exists also in the SFA implementation of Planetlab Europe.

A simple credential [82]might look like this:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<signed-credential>
 <credential xml:id="ref1">
```

```
  <type>privilege</type>
  <serial>12345</serial>
  <owner_urn>urn:publicid:IDN+emulab.net+user+gary</owner_urn>

<target_urn>urn:publicid:IDN+emulab.net+slice+mytestslice</target_u
rn>
  <expires>2010-01-01T00:00:00</expires>
  <privileges>
   <privilege>
    <name>*</name>
      <can_delegate>1</can_delegate>
   </privilege>
  </privileges>
 </credential>
 <signatures>
  ---XML Signature elements (signatures and certificates) go here--
-
 </signatures>
</signed-credential>
```

In the case of the SFA credentials we can derive the affiliation of the user by the HRN (e.g. “topdomain.subdomain.username”). This describes a chain of trust where the topdomain signs the certificate of the subdomain, which in turn signs the certificate of the user. The “A/A” can validate the user’s credentials by verifying the signatures and the certificates in a chained way.
The user proves itself with assertions included in his initial request (e.g., request to list resources available on the testbed). The assertions are of the type [subject, predicate, object] (e.g. DonatosMemberOf UTH says UTH-HQ). The A/A can simply check the user's signature with the corresponding authority (e.g. UTH-HQ). The assertions can be tracked in the following chain way:

```
1. MSG isType REQ from ?user
2. ?user memberOf ?project says ?pa
3. ?pa endorsedBy OpenLab says OpenLabAuth.
```

This way, the A/A can make a decision whether to deny or grant access to a user’s REQ.

Scheduler

The Scheduler receives a request for resources reservation within a specific timeframe and decides how to match the user's request. At the moment, it supports Hard Reservations with regards to time (as shown in Table 11). Hence it simply checks the availability of the resources in the given timeframe and if they are available, it reserves them on behalf of the user for a slice. More complex examples include user/testbed imposed constraints such as minimum duration, preferable duration, minimum resources, and preferable resources. In addition it could prioritize requests based on user information/privileges. Following the selection of resources and the storing of the reservation information in the DB, the Scheduler informs the AM Liaison about the new reservation in order to schedule a provisioning event before the reservation starting time. Finally, the Broker informs the user about the result of his request.

The following Table 13 provides an example of a request to reserve a node from 2013/1/8 19:00 till 2013/1/8 20:00

```
<?xml version="1.0"?>
<rspec xmlns="http://www.protogeni.net/resources/rspec/2"
xmlns:omf="http://schema.mytestbed.net/sfa/rspec/1" type="request"
xmlns:olx=" http://schema.nitlab.inf.uth.gr/sfa/rspec/1">
  <olx:lease lease_name="l1" olx:valid_from="2013-01-08T19:00:00Z"
olx:valid_until="2013-01-08T20:00:00Z"/>
  <node component_id="urn:publicid:IDN+openlab+node+node1"
component_name="node1" olx:lease_name="l1">
  </node>
</rspec>
```

Table 13: Node Reservation Request Example

The following is the confirmation that your lease is accepted and is assigned with a uuid for future reference.

```
<?xml version="1.0"?>
<rspec xmlns="http://www.protogeni.net/resources/rspec/2"
xmlns:omf="http://schema.mytestbed.net/sfa/rspec/1"
type="advertisement" xmlns:olx="
http://schema.nitlab.inf.uth.gr/sfa/rspec/1">
  <olx:lease lease_uuid="550e8400-e29b-41d4-a716-446655440000"
olx:valid_from="2013-01-08T19:00:00Z" olx:valid_until="2013-01-
08T20:00:00Z"/>
  <node component_id="urn:publicid:IDN+openlab+node+node1"
component_name="node1" olx:lease_uuid="550e8400-e29b-41d4-a716-
446655440000"/>
  </rspec>
```

Table 14: Node Reservation Confirmation Example

AM Liaison

The AM Liaison is responsible for keeping in sync the scheduler database with all the available resources from the AM(s) and serves all the resource allocation requests. Two types of communication methods are defined:

1. Via the SFA API in the case of SFA-enabled testbeds. RSpec PG v2 format is used in order to communicate with the AM.

2. OMF messaging system which is based on the XMPP protocol in order to send “create” messages to the corresponding AMs whenever a reservation is about to start.

Last but not least, the Liaison can send requests to an underlying Broker (hierarchical architecture) using one of the available interfaces (XMPP or SFA).

The schema of the database is being generated automatically by the usage of ORM (Object-relational mapping). The main objects that constitute the core of the information model are the following:

- OResource
- OProperty
- OAccount
- OComponent
- OLease

Figure 9 depicts the basic components of the information model:

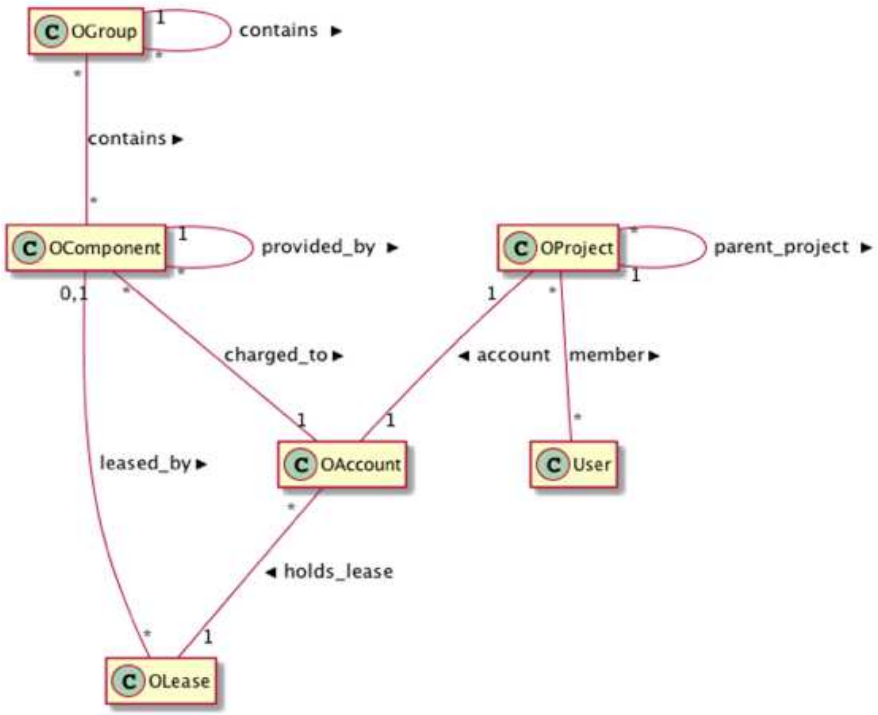

Figure 9: Components of Reservation Information Model

"OResource" is the general class for describing a resource along with the "OProperty" class. All the resources inherit the "OResource" class like the "OLease" which is a resource with its own properties. The "OAccount" and the "OComponent" also inherit the "OResource" class. More specific the "OResource" class contains the following information:

- id (Serial)
- type (Class type e.g. OAccount, OLease)
- uuid (UUID)
- name (String)
- urn (String)
- resource_type (String)
- has n Properties (This way we declare that an object of the class "OResource" has multiple objects of the class "OProperty")
- belongs to an OAccount (This way we describe the relationship between the "OResource" class and the "OAccount" class.

"OProperty" class contains the following attributes:

- id (Serial)
- name (String)
- value (String)
- belongs to a OResource (This way we declare that an object of the class "OProperty" belongs to an object of the class "OResource")

"OComponent" class describes the resources that have a management interface. All the physical resources are considered OComponents and whenever there is a new resource it should extend this class. The attributes of the class are the following:

- domain (String)
- exclusive (Boolean)
- status (String)
- provides n times its self (This helps describing the virtualization of a resource. Even when the resource is not virtualized, conceptually the experimenter is provided a clone of the resource)
- has n OLeases (This way we describe that a "OComponent" can have multiple reservations "OLease" objects.)

"OLease" is the class which describes the reservations with the following attributes:

- valid_from (DateTime)
- valid_until (DateTime)
- has n OComponents (This way we declare that an object "OLease" can have multiple instances of the class "OComponent")

Here is an example of adding a "Node" resource to the above information model. All you have to do, is to extend the class "OComponent" and add your properties ("OProperty"):

```
class Node < OComponent
      oproperty :hardware_type, String, :required => false
      oproperty :available, Boolean, :default => true
      oproperty :interfaces, :interface, :functional => false
":functional => false" declares that the property has an array of
values.
```

Hierarchical usage of the Broker

The NITOS broker can also be used in a hierarchical reservation system where a main Broker can be contacted directly from a client and reserve resources that have been assigned to the main Broker by another (slave) Broker.

In a scenario where the hierarchical architecture is being used (a main Broker and a slave Broker), the messages are being forwarded from the main Broker to the slave Broker intact. If a user requests an advertisement of the resources, then the main Broker is able to provide that without contacting the slave Broker, because it has already all the available resources of the slave Broker

in its database (in particular the resources that the slave Broker provided for brokering to the main Broker).

Upon receiving a valid request from the experimenter, the main Broker serves this request by making an SFA call to the underlying Broker forwarding the user's RSpec. The slave Broker is configured to accept all requests from the master Broker if it includes resources that have been allocated to the latter. In this way both Brokers maintain the same information in their databases regarding the resources the user has requested. The manifests generated on this two level reservation system are identical, therefore the master Broker returns the manifest created at its level without the need to extract the manifest from the slave Broker.

In order to support hierarchical scheduling, the slave Broker(s) must accept all requests from the "master" Broker. When a testbed owner decides to assign a subset of the testbed's resources for brokering to the "master" Broker, there is a need to populate the database of the "master" Broker with this subset. The "master" should have the same information model with the "slave" Broker (s). The population of the database is manual for the moment.

In Figure 10 a simple deployment of the Broker on a testbed which is SFA enabled is depicted

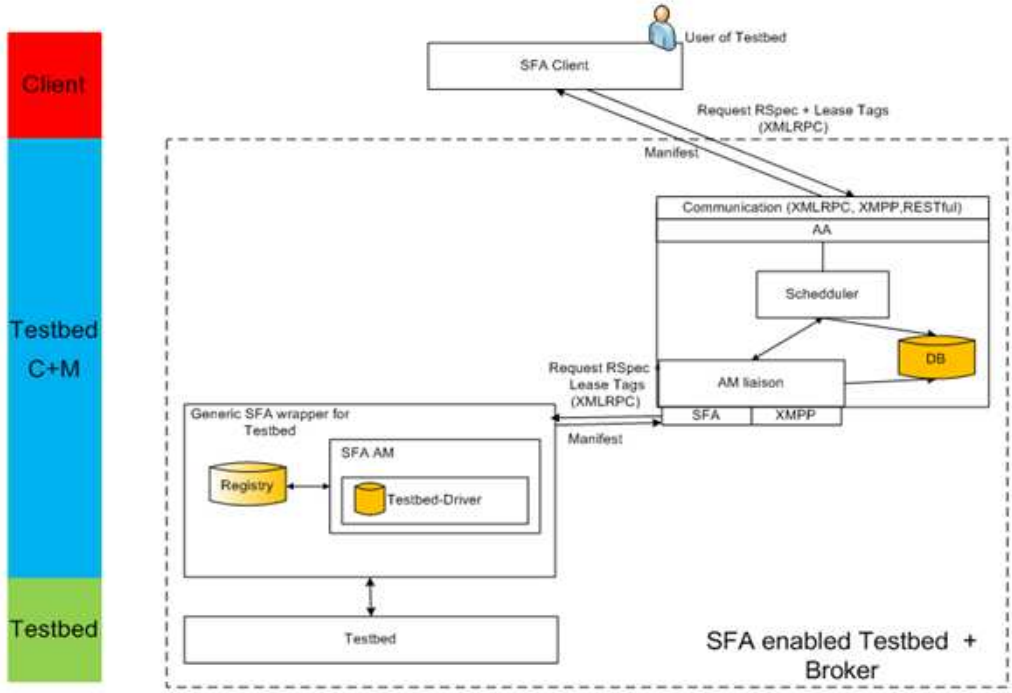

Figure 10: Broker inside an SFA-enabled testbed

Figure 11 illustrates the hierarchical use of the master Broker on this testbed.

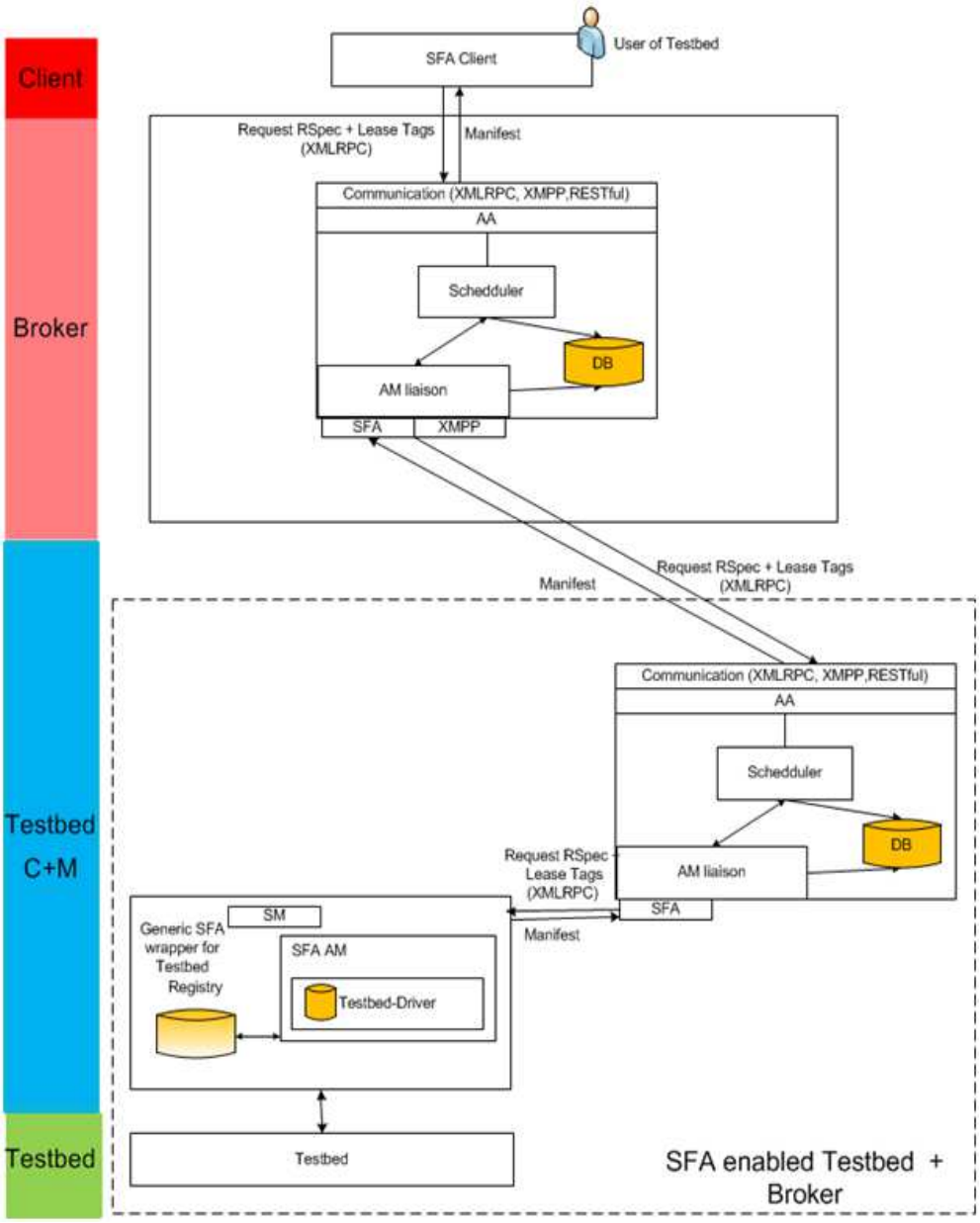

Figure 11: Hierarchical use of the Broker

Implementation Status

So far, the XML-RPC interface is implemented along with the AA component (currently under testing). The RSpec parsing mechanism and the information model of the DB are fully functional. The Scheduler has a basic functionality for reserving and releasing resources based on a calendar service.

3.5.4 Required additional implementations

The Broker will have ready the following functionalities before the end of cycle 1, which will allow for its initial deployment. Beyond the XML-RPC interface the following two interfaces XMPP and REST will be finished so that OMF testbeds will be able to adopt the Broker without any further effort as it will be provided out of the box.

The extension of NITOS Scheduler as part of the Broker will be further enhanced with more scheduling algorithms beyond the "First Come First Served" simple policy. Furthermore, anyone will be able to introduce his own custom made conflict resolution algorithm for reservations.
Also, the implementation of the AM Liaison will provide the connection with the reserved resources. AM Liaison is the component responsible to send any provisioning commands to the resources or to their component managers. The broker's interfaces will be consistent with the XMPP OMF protocol and the SFA API.
A plugin for the portal also needs to be developed that acts as the front-end of the deployed Fed4FIRE Future Reservation Broker. This plugin has to be able to indicate the availability of resources over time, and has to allow experimenters to request the different types of reservations that are supported by the Broker.
Finally, the implemented Future Reservation Broker will also be packaged in such a way that it can easily be used by testbeds as their local reservation engine. This means that testbeds that do not yet provide reservation support in their testbed management software will be able to easily include this by installing this specific version of the Future Reservation Broker locally. This can be of particular interest to testbeds which do not yet support reservations, but which also do not want to outsource this task to the central Fed4FIRE Future Reservation Broker. In this specific packaging of the Broker there will be the need to omit specific functionalities in order not to conflict with the existing testbed management software. However, some more analysis and tests should be performed in the context of this use case. Therefore it is considered to be too preliminary to define the final specifications of this particular Broker package right now.

In summary the functionalities implemented by the end of cycle 1 are listed in Table 15:

Communication Interfaces	XMPP XML-RPC (SFA) REST
Authentication and Authorization	Mapping SFA credentials to native Broker's permissions
Scheduler	Simple "First Come First Served" implementation for serving reservation requests
AM Liaison	XMPP interface for controlling OMF6 RCs
Database	The basic information model described in section 3.4
Testbed-local flavour	Testbeds will be able to use a specific flavour of the Broker as a local reservation engine for their testbed.

Table 15: Resource reservation functionalities to be implemented for cycle 1

3.5.5 Specifications

1) Resource Description

An appropriate resource description format will be adopted, as dictated by Task 5.2, to facilitate resource reservation for request types presented in section 3.5.1, for every testbed in the federation. As an initial step, the XML-based resource specification language RSpec (Protogeni v2) has been extended to support In advance reservations. Specifically the RSpec request schema has been extended with the element "lease"; an example of a reservation as such can be seen below:

```
<?xml version="1.0"?>
<rspec xmlns="http://www.protogeni.net/resources/rspec/2"
xmlns:omf="http://schema.mytestbed.net/sfa/rspec/1" type="request"
xmlns:olx="http://schema.nitlab.inf.uth.gr/sfa/rspec/1">
  <olx:lease lease_name="l1" olx:valid_from="2013-01-08T19:00:00Z"
olx:valid_until="2013-01-08T21:00:00Z"/>
  <node component_id="urn:publicid:IDN+openlab+node+node1"
component_name="node1"
 olx:lease_name="l1">
  </node>
</rspec>
```

The lease extension (in the form of .xsd) can be seen below:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:olx="
http://schema.nitlab.inf.uth.gr/sfa/rspec/1"
elementFormDefault="qualified"
targetNamespace="http://schema.nitlab.inf.uth.gr/sfa/rspec/1">
  <xs:element name="lease">
      <xs:complexType>
      <xs:attribute name="lease_name"/>
      <xs:attribute name="lease_uuid"/>
      <xs:attribute name="valid_from" use="required"/>
      <xs:attribute name="valid_until" use="required"/>
      </xs:complexType>
  </xs:element>
</xs:schema>
```

2) Communication Interfaces

The SFA API [25] serves as the main API for enabling federation among the heterogeneous testbeds of Fed4FIRE. Furthermore it enables the hierarchical usage of the Broker. The Broker's SFA API is described hereafter.

API

- **GetVersion**
  - Get static version and configuration information about this aggregate.
- **ListResources**
  - Return a listing and description of available resources at this aggregate, or resources allocated to a named slice at this aggregate.

- **CreateSliver**
  - Allocate resources as described in a request RSpec argument to a slice with the named URN.
- **DeleteSliver**
  - Delete any slivers at the given aggregate belonging to the given slice, by stopping the resources if they are still running, and then de-allocating the resources associated with the slice.
- **SliverStatus**
  - Get the status of a sliver or slivers belonging to the given slice at the given aggregate.
- **RenewSliver**
  - Renews the resources in all slivers at this aggregate belonging to the given slice until the given time, extending the lifetime of the slice.
- **Shutdown**
  - Perform an emergency shut down of a sliver or slivers at this aggregate belonging to the given slice.

The XMPP interface of the broker leverages the new OMF 6 messaging protocol [OMF6] which consists of the following fundamental messages:

- Create Message
- Configure Message
- Request Message
- Release Message
- Inform Message

3) Scheduler

The scheduler should cater for in advance reservations and simple “First Come First Served” policy.

4) Database

The database of the master reservation broker should be synchronized with the slave broker.

5) AM Liaison

The master broker should be able to communicate via the appropriate protocol (XMPP, SFA) with the underlying AMs.

3.5.6 Requirements for testbeds to interact with the Future Reservation broker

Testbeds aiming to support this functionality should comply with the following requirements:

- SFA-enabled with a scheduling solution, or
- SFA-enabled without a scheduling solution

For the first case, in order to comply with the hierarchical scheme, the testbed's existing scheduler needs to act as a slave broker for the Future Reservation Broker (master).
In the latter case, the testbed must export a pre-selected set of testbed resources to the Future Reservation Broker, or deploy a local scheduling solution which can act as a slave broker for the Future Reservation Broker. In the latter case a possibility is to apply the specific package of the Future Reservation Broker that is intended to be used as a local reservation engine.

3.6 Exposing testbeds through SFA

3.6.1 General description

SFA defines the minimal set of interfaces and data types that permits a federation of slice-based network components to inter operate. Component owners declare policies for resource allocation and usage on the network under their control. Users allocate and access resources across the entire network. In D2.1 (First Federation Architecture) it was decided that all Fed4FIRE testbeds should be exposed through SFA. Main drivers behind this decision were the fact that the SFA framework complies with all the imposed architectural requirements, and the fact that the majority of the Fed4FIRE testbeds already had SFA support on their own roadmaps. Therefore, the main idea is that we leave testbeds the freedom to choose how they want to implement this SFA interface on top of their testbed management systems. However, for testbeds who have not yet dug deeper into the SFA route, WP5 should provide generic tools that allow them to also adopt SFA with limited efforts and in a relatively short amount of time. In this section we will define which tools will be provided in that case.

Unlike the other sections, this does not present a tool that serves as a solution for a demanded functionality. Here is presented a solution that each testbed must adopt in order to achieve the FED4FIRE requirements. So, the particular technical specification must be decided by each testbed.

3.6.2 Evaluation of possible approaches for implementation

As explained in the previous section, SFA provides a common interface to expose testbed resources to the outside world. Several approaches can be adopted when implementing this interface for a given testbed:

- SFA Aggregate Manager / SFA (GENI [61], PlanetLab [33], AMsoil [62], genericSFA [1]) – (i2CAT,INRA)
- OMF Provisioning [63] - (NICTA)
- Emulab [67]/ Virtual Wall [34] (iMinds)
- Teagle Framework [68] / Orchestration Engine and PTM [69]

Table 16 and Table 17 summarize the main characteristics of these different solutions. The evaluated criteria are the following:

- **Supported functions:** Here the basic supported functionalities of the tool are listed.
- **APIs:** The APIs that the tool can communicate with.
- **Extensibility:** The analysis on how the tool can be extended.
- **Orch. Functionality:** Here it is described if the tool has orchestration functionality and how it works if so.
- **Resource description:** A brief explanation of how the resources are described through the tool.
- **Arch. Specification:** Corresponds to a short description of the current architecture of the tool.
- **Prog. Lang:** The programming language in which the tool was built.

- **Supported standards:** The different standards, which the tool is capable to work with.
- **Where is used:** Where the tool is used, in which testbed or scenario.
- **Licenses:** Under what license the tool is distributed.

Properties	AMsoil	Generic SFA	OMF Provisioning
Supported functions	Authorization and authentication, database, policies (pyPElib), monitoring. Slice CRUD.	Full SFA API: Authentication, Authorization, Policies (sfatables) Additional Components: Scheduler, UI, Monitoring…	Resources CRUD and other functionalities depending on the resource type
APIs	GENI/SFA, OCF	GENI, SFA	RESTful (REST-based client), XML-RPC (SFA Client), XMPP (OMF-based client)
Extensibility	Extensible through plugins	Extensible through drivers	Testbed-specific provisioning code can be added as another service module software.
Orch. functionality	No		No (maybe in future versions)
Resource description.	GENI v1 for OpenFlow OCF RSpec for VMs		OEDL
Arch. Specificities	Skeleton code for AM implementation.	Generic class for AM Testbed specific code in a driver	Provisioning implemented as ‘services’ provided by an OMF Aggregate Manager.
Prog. Lang.	Python	Python	Ruby
Supported	GENI SFA v1	SFA v1, ProtoGeni v2	XML-RPC API

Properties	AMsoil	Generic SFA	OMF Provisioning
standards		& v3	provided by the OMF Provisioning complies with the SFA Client API designed by GENI
Where is used	OFELIA, FIBRE and EXPERIMENTA testbed	PlanetLab, Nitos , Federica, SensLab (Fit, F-Lab, OpenLab, Fibre, NOVI)	NITOS, INRIA, PlanetLab, NICTA, Winlab,
Licenses	GitHub licensing policy.	Not applicable.	MIT license

Table 16: Comparision of eligible tools for exposing testbeds through SFA – part 1

Properties	Emulab / Virtual Wall – GENI SFA	Teagle FW / Orchest. Eng.
Supported functions	Slice CRUD, Start/Stop, information; Authorization and access control	CRUD, Start/Stop, AA, Persistence
APIs	GENI API, The GENI AM API is intended to be compatible with the SFA	SFA AM, Database REST + GUI, PTM/ReqProcessor REST + CLI + GUI, RA REST + GUI, OMA PolicyEngine
Extensibility	GENI defines the Aggregate Manager API (SFA), but the AM can drive all kinds of sources	Extensible through arbitrary Resource Adapters (RAs)
Orch. functionality	depends on the AM, but e.g. the AM of Emulab can orchestrate questions as '5 machines with this topology'	Dedicated Orchestration Engine, capable of resolving dependencies and orchestrating provisioning requests across multiple testbeds and resources
Resource description.	GENI RSpec, OpenFlow RSpec	Resource Adapter Description Language (RADL)
Arch. Specificities	SFA arch	Centralized database and orchestration, distributed PTMs/Ras (tree model)
Prog. Lang.	Python, XMLRPC, flash, ...	Python, Java, Groovy

Properties	Emulab / Virtual Wall – GENI SFA	Teagle FW / Orchest. Eng.
Supported standards	GENI	Open Mobile Alliance (OMA) Policy Evaluation, Enforcement, and Management (PEEM)
Where is used	GENI	Panlab, Fraunhofer, TUB, UoP
Licenses	GENI License policy.	Apache 2.0

Table 17: Comparison of eligible tools for exposing testbeds through SFA – part 2

Armed with this knowledge, it is now possible to analyze which of the different technical solutions would seem to be the most appropriate to cover the needs in Fed4FIRE of exposing all testbeds through SFA.

The first option is to implement this **from scratch**. This is however not the optimal solution since this requires large investments of manpower, while solutions already exist that could greatly reduce the amount or needed work.

A second solution is to replace the existing testbed management system by one that already exposes itself through SFA. One possibility could be to **deploy Emulab** [67] on the testbed. This testbed management software framework is able to expose reserve and provision nodes, and has a proven track record of providing all this functionality through SFA. This is a technically feasible solution, but this feasibility depends on the specific characteristics of the testbed. If its current testbed management system provides functionalities that are not provided by Emulab, then additional implementations are needed to guarantee that there is no loss in functionality because of the switch of management system. Therefore it is not considered to be a good generic all round solution that can be provided by WP5 to any testbed that is new in the SFA world.

Another existing testbed management software framework that already supports SFA in production is that of **PlanetLab Europe** [49]. However, this is again a testbed-specific implementation, while the Generic SFA Wrapper [60] is in fact the generalized version of this (originally) PlanetLab Europe implementation. Therefore it does not make sense to focus on the specific PLE testbed management software in this context.

A third testbed management software framework that supports SFA is **FITeagle** [55]. However, the SFA capabilities of this framework were in fact implemented using the Generic SFA Wrapper. Therefore, just as for the PlanetLab Europe case described above, it again wouldn't make sense to focus on the testbed-specific form (in this case FITeagle) in this context.

The fourth testbed that can be considered is **OMF** [63], since it promises to provide SFA support natively in one of its next releases. This solution is however characterized by the same disadvantages as the Emulab solution: when replacing the existing testbed management software with OMF, it cannot be excluded that quite some efforts will be needed in order to maintain the already provided level of functionalities. So it is just not generic enough to be considered in this context.

When focusing on generic solutions that can assist testbeds in their quick and seamless transition to the SFA domain, two possibilities remain: the Generic SFA Wrapper, and the AMsoil solution.

As mentioned before, the **Generic SFA Wrapper** can be introduced as the generalized implementation of the SFA interface on top of PlanetLab Europe. This functionality has already proven to be mature, and is now packaged in such a way that it can be used relatively easily to wrap an existing testbed management framework with an SFA interface. What the wrapper basically does is providing a finished implementation of the SFA side, and providing some empty stubs at the testbed side. This is called the testbed driver. So testbeds just have to fill in these stubs with appropriate calls to their existing testbed management software in order to implement SFA support for their testbed. So this is a mature and generic solution to expose any testbed through SFA with as little efforts as needed. **The Generic SFA Wrapper is therefore considered as one of the candidate tools that will be provided by WP5 to the testbeds.**

The other solution, **AMSoil**, is currently being developed in the context of the FP7 OFELIA project. In this project, a complete redesign of the existing testbed management framework is currently being implemented. Native SFA support is one of the key characteristics of the new design. It is currently being developed in a two-phased approach: first a generic testbed management framework including SFA support is being written, and then it is adapted and extended in order to be capable of managing the Ofelia testbeds. The intention is that the generic framework already provides the majority of the needed functionality, and that the step towards the actual testbed is rather small. One could however do the same for other testbeds: replace the existing management software by one based on AMSoil, where only limited adaptations have to be implemented to support the specific testbed. So instead of wrapping your testbed, your are replacing it by a generic, SFA supporting testbed management software framework. Although this solution is not yet as mature as the Generic SFA Wrapper, it is considered to be a valuable alternative approach that testbeds might warm up to. **Therefore the AMSoil approach is the second candidate that is considered to be suitable to be provided to the testbeds by WP5.**

Table 18 summarizes the previous conclusions:

Approach	Advantages	Disadvantages	Selected as final approach
Implement from scratch	Covers all requirements.	A lot of effort needed. Other solutions exist that could lead to less needed efforts.	
Replace specific testbed management system by Emulab	Covers all requirements. Mature implementation of the SFA interface.	Not generic: some cases might require a lot of effort.	
Replace specific testbed management	Covers all requirements. Mature implementation of the SFA interface.	Not generic: some cases might require a lot of effort.	

Approach	Advantages	Disadvantages	Selected as final approach
system by PLE		Generic SFA wrap is the generic derivative of this implementation, so no incentive to rely on the PLE specific implementation.	
Replace specific testbed management system by FITeagle	Covers all requirements. SFA part already provided.	Not generic: some cases might require a lot of efforts. This is implemented using the Generic SFA Wrapper, so no incentive to rely on the FITeagle specific implementation.	
Replace specific testbed management system by new OMF release	Covers all requirements.	Not generic: some cases might require a lot of efforts.	
Use Generic SFA Wrapper to create an SFA interface on top of the existing testbed management framework.	Covers all requirements. Mature implementation of SFA. Generic solution → minimal efforts needed at testbed side.	Additional management layer instead of introducing native SFA support → additional maintenance efforts might be needed on the long term.	X
Use AMSoil to replace the current testbed management framework by a generic (extensible) one that natively supports SFA	Covers all requirements. Native solution → easier to maintain. Generic solution → minimal efforts needed at testbed side.	Not yet mature.	X

Table 18: Evaluation of possible approaches for enabling SFA across Fed4FIRE's testbeds

3.6.3 Description of selected tools

After considering all the tools elaborated upon in the previous section, we consider two ways to provide an SFA API to the testbeds. Both are based on tools that can be easily adopted in order to make a testbed fully SFA compliant. The suggested tools are AMsoil [62] and SFAWrap [60]. Both tools will be described in more detail in following sections. There are no clear advantages or disadvantages between both tools, although they cover different needs, so **the decision of which one to choose is more based on the actual needs of each testbed**.

As mentioned before, the Generic SFA Wrapper can be introduced as the generalized implementation of the SFA interface on top of PlanetLab Europe. This functionality has already proven to be mature, and is now packaged in such a way that it can be used relatively easily to wrap an existing testbed management framework with an SFA interface. What the wrapper basically does is providing a finished implementation of the SFA side, and providing some empty stubs at the testbed side. This is called the testbed driver. So testbeds just have to fill in these stubs with appropriate calls to their existing testbed management software in order to implement SFA support for their testbed. So this is a mature and generic solution to expose any testbed through SFA with as little efforts as needed. It is therefore considered as one of the candidate tools that will be provided by WP5 to the testbeds. Since the Generic SFA Wrapper is already in use at several testbeds, **we consider the provisioning of the Generic SFA Wrapper from WP5 to all interested testbeds feasible within cycle 1.**

The other solution, AMSoil, is currently being developed in the context of the Ofelia project. In this project, a complete redesign of the existing testbed management framework is currently being implemented. Native SFA support is one of the key characteristics of the new design. It is currently being developed in a two-phased approach: first a generic testbed management framework including SFA support is being written, and then it is adapted/extended in order to be capable of managing the Ofelia testbeds. The intention is that the generic framework already provides the majority of the needed functionality, and that the step towards the actual testbed is rather small. One could however do the same for other testbeds: replace the existing management software by one based on AMSoil, where only limited adaptations have to be implemented to support the specific testbed. So instead of wrapping your testbed, you are replacing it by a generic, SFA supporting testbed management software framework. Although this solution is not yet as mature as the Generic SFA Wrapper, it is considered to be a valuable alternative approach that testbeds might warm up to. Therefore the AMSoil approach is the second candidate that is still considered to be a suitable mechanism to be provided to the testbeds by WP5. However, based on the current implementation status, **AMSoil is not considered to be ready for deployment on the testbeds during cycle 1**.

Description of SFAWrap

Overview

SFAWrap is one of the most visible and renowned reference implementation of the Slice-based Federation Architecture (SFA) [1], the emerging standard for networking experimental testbed federation. Put on top of testbed's control and management frameworks, SFAWrap provides

those testbeds with SFA compatibility allowing them to expose their resources to the federation community.

SFAWrap defines the minimal set of interfaces and data types that permit a federation of slice-based network components to interoperate. Component owners declare policies for resource allocation and usage on the network under their control. Users allocate and access resources across the entire network.

Slices are the primary abstraction for accounting and accountability: resources are acquired and consumed by slices, and external program behaviour is traceable to a slice. And, a slice is defined by the set of resources spanning a set of network components, plus an associated set of users that are allowed to access the resources for the purpose of running an experiment. By formalizing the interface around the slice, resource owners and users are free to cooperate more easily. Owners simplify the administrative overhead of making their systems easily accessible to more users, and users gain access to interesting systems without the overhead of setup and administration.

SFAWrap Components:

The following Figure 12 depicts the main components SFAWrap overall architecture.

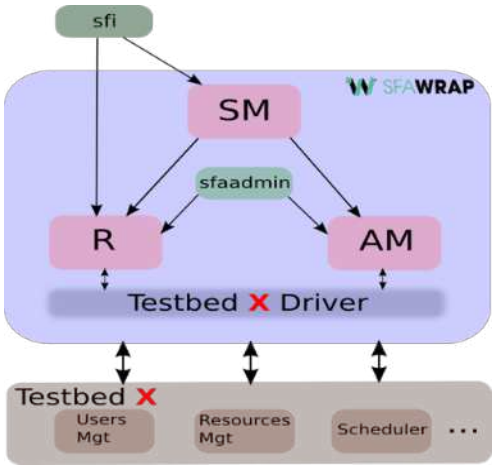

Figure 12: Overall architecture of SFAWrap (R: Registry, AM: Aggregate Manager, SM: Slice Manager)

Registry (R):

The Registry is an XMLRPC over HTTPS service that exports exactly the Registry API. The Registry is responsible for maintaining and serving SFA records namely: Authorities, Users and Slices, and also issues the related certificates and credentials.

The Registry can be deployed in a standalone mode in order to be used only to issue user and slice credentials.

Aggregate Manager (AM):

The Aggregate Manager is an XMLRPC over HTTPS service that exports exactly the Aggregate Manager API. The Aggregate Manager is responsible for performing all the slice instantiations and also, allowing testbed aggregates to advertise their resources and attach those latter to slices.

Slice Manager (SM):

The Slice Manager is an XMLRPC over HTTPS service that exports also the Aggregate Manager API, but that has no real testbed attached directly. Instead, it acts as a proxy that is aware of a pre-configured set of other services, either Aggregate Manager or in turn Slice Manager.

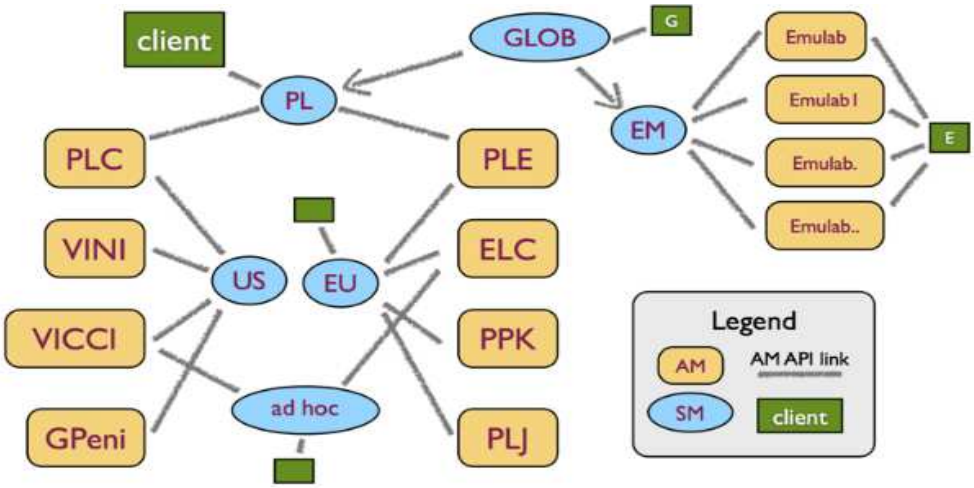

Figure 13: The Slice Manager and its function in the federation

This means that a given client would essentially need (to be configured with) only one single entry in the federation, and then discovers a portion of the overall picture that depends on that entry point. As depicted in Figure 13, the Slice Manager will be in our opinion a very convenient option to master complexity over time and scale. It is also a very good illustration of the decentralized nature of the SFA infrastructure.

Testbed Driver:

The testbed driver is the part of SFAWrap that deals with the testbed specificities and talks to the testbed management framework. Depending on how users and resources are managed within the testbed itself, the driver will need to translate the Aggregate Manager API and the Registry API methods in order to match respectively with the testbed resources allocation/provisioning and the testbed users management and access policies.

Below is the UML diagram of the 'driver' class:

DRIVER
+ hrn: String
+ augment_records_with_testbed_info() + register() + remove() + update() + update_relation()

+ testbed_name()
+ aggregate_version()
+ list_slices()
+ list_resources()
+ sliver_status()
+ create_sliver()
+ delete_sliver()
+ renew_sliver()

More developer oriented details are depicted below in the abstract 'driver' class implementation :

DRIVER CLASS

```
class Driver:

    def __init__ (self, config):
        # this is the hrn attached to the running server
        self.hrn = config.SFA_INTERFACE_HRN

    ########################################
    ##########  registry oriented  #########
    ########################################

    # the following is used in Resolve (registry) when run in full mode
    # after looking up the sfa db, we wish to be able to display
    # testbed-specific info as well

    def augment_records_with_testbed_info (self, sfa_records):
        return sfa_records

    # incoming record, as provided by the client to the Register API
call
    # expected retcod 'pointer'
    # 'pointer' is typically an int db id, that makes sense in the
testbed
    # environment
    # -1 if this feature is not relevant

    def register (self, sfa_record, hrn, pub_key) :
        return -1

    # incoming record is the existing sfa_record
    # expected retcod boolean, error message logged if result is False

    def remove (self, sfa_record):
        return True

    # incoming are the sfa_record:
    # (*) old_sfa_record is what we have in the db for that hrn
```

```
    # (*) new_sfa_record is what was passed in the Update call
    # expected retcod boolean, error message logged if result is False
    #
    # NOTE 1. about keys
    # user may have several ssh keys on the testbed but we need to pick
    # one to generate its cert, the manager code actually picks one
(the
    # first one), and it seems safer to pass it along rather than
    # depending on the driver code to do the same
    #
    # NOTE 2. about keys
    # when changing the ssh key through this method the gid gets
changed
    # too

    def update (self, old_sfa_record, new_sfa_record, hrn, new_key):
        return True

    # callack for register/update
    # this allows to capture changes in the relations between objects
    # the ids below are the ones found in the 'pointer' field
    # this can get typically called with
    # 'slice' 'user' 'researcher' slice_id user_ids
    # 'authority' 'user' 'pi' authority_id user_ids

    def update_relation (self, subject_type, target_type
,relation_name,
                         subject_id, link_ids):
        pass

    ########################################
    ##########  aggregate oriented  ########
    ########################################

    # a name for identifying the kind of testbed

    def testbed_name (self): return "undefined"

    # a dictionary that gets appended to the generic answer to
GetVersion
    # 'geni_request_rspec_versions' and 'geni_ad_rspec_versions' are
    # mandatory

            def aggregate_version (self): return {}

    # the answer to ListSlices, a list of slice urns

    def list_slices (self, creds, options):
        return []

    # answer to ListResources
```

```
    # first 2 args are None in case of resource discovery
    # expected : rspec (xml string)

    def list_resources (self, slice_urn, slice_hrn, creds, options):
        return "dummy Driver.list_resources needs to be redefined"

    # the answer to SliverStatus on a given slice

    def sliver_status (self, slice_urn, slice_hrn):
          return {}

    # the answer to CreateSliver on a given slice
    # expected to return a valid rspec
    # identical to ListResources after the slice was modified

    def create_sliver (self, slice_urn, slice_hrn, creds, rspec_string,
                       users, options):
        return "dummy Driver.create_sliver needs to be redefined"

    # the answer to DeleteSliver on a given slice

    def delete_sliver (self, slice_urn, slice_hrn, creds, options):
        return "dummy Driver.delete_sliver needs to be redefined"

            # the answer to RenewSliver
            # expected to return a boolean to indicate success

    def renew_sliver (self, slice_urn, slice_hrn, creds,
expiration_time, options):
        return False
```

SFAADMIN

"sfaadmin" is a command line tool that allows administrator and operators to perform Slice or Registry operations (via the Aggregate Manager API or the Registry API) within an admin context without the use of credentials.

SFI

"sfi" is a user-oriented command line tool that manages a set of credentials on behalf of the user, and uses them when performing Slice or Registry operations (via the Aggregate Manager API or the Registry API).

RSpecs

Testbed resources within SFAWrap, which are heterogeneous in the context of federation, are described using RSpec (Resource Specification). The RSpec exposes the resources in terms of : Type, Capabilities, Provisioning Policy and Access Policy. It also depicts the testbed in terms of time-based resource reservation in order to advertise about the availability of those resources for a given timeslot.

Currently, SFAWrap supports three versions of RSpecs, namely:

- PlanetLab RSpecs in three different implementation: SFAv1, PROTOGENIv2 and GENIv3.
- NITOS RSpecs.
- SensLab RSpecs.

SFAWrap Implementation

Initially SFAWrap code was targeting PlanetLab testbeds only. With this in mind, the whole code was redesigned to reach the design that is depicted in Figure 14. On this picture, the dark pink boxes represent the core of the generic wrapper; the light gray boxes represent the pieces of code that implement the 'plugin' system; and the dark blue boxes represent the testbed-specific code that needs to be written in order to provide an implementation.

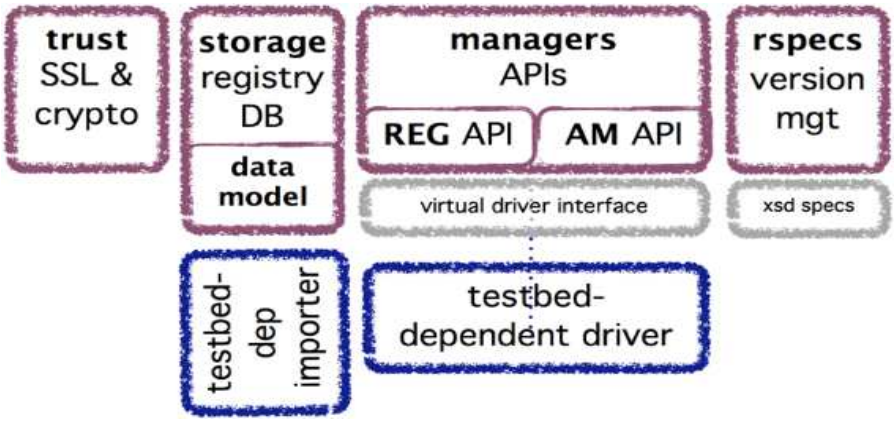

Figure 14: SFAWrap design

In more details, and as depicted in Figure 14 the various parts of the generic code [46] are:

trust: This package implements all the gory details related to SSL certificates, GIDs, and hierarchy management in terms of the chain of trust; this of course could also be used as a standalone library if the need arises.

storage: This package implements the data model underlying all the entities in the registry system, and namely Authorities, Users and Slices. As an implementation note, let us stress that we have taken advantage of the sqlalchemy library [48], which allowed us to very easily add relationships into the model, that formerly only involved a flat table of records. As a matter of fact, the registry needs to know at least which users belong in which authority, and which of them are allowed to act as this authority (in other words, have PI authorization), as well as which users are in a given slice.

managers: This package provides a reference implementation of the 3 available services, namely the Aggregate Manager, Registry and Slice Manager. Although the last one does in essence not exhibit any relationship to a given testbed (being exposed only to SFA entities), the first two do strongly depend on the testbed being wrapped. This is where the notion of a testbed driver comes in. The testbed driver is expected to fulfil the already defined virtual interface. A configuration mechanism then allows selecting the driver to use at runtime.

rspecs: This package provides an abstraction in order to manipulate resource descriptions in a way that does not depend too much on the version of the RSpec formalism being used (as for legacy, clients have the option to choose among several formalisms). There is also a provision for testbed operators to provide their specific RSpec formalism(s) in the XSD format.

More hands-on information about how to use the system from a programmer's point of view can be found at [47].

SFAWrap Flavours

Currently, SFAWrap is used in six different flavours, in other words, SFAWrap is running on top of six heterogeneous testbeds and providing them with a common control plane interface.

Those six flavours are:

- PlanetLab Europe [49] (UPMC/INRIA - France)
- NITOS [51] (CERTH - Greece)
- SensLab [52] (INRIA - France)
- Federica [53] (i2CAT - Spain)
- FITeagle [55] (TUB - Germany)
- OSIMS [56] (University Of Patras - Greece)

3.6.4 Specifications for SFAWrap

SFAWrap API's:

SFAWrap implements the Aggregate Manager API and the Registry API.

Aggregate Manager API

The Aggregate Manager API focuses on defining the primitives that an Aggregate Manager (AM) which in the SFA jargon refers to a testbed management infrastructure - has to provide to advertise resources and to allocate resources to Slices in order to be SFA-compliant.

The Aggregate Manager API is the control plane interface by which experimenters discover, reserve and control resources at resource providers. It does not include resource specific interactions, application level interactions, or monitoring and management functions.

Currently, SFAWrap fully supports AM API v2 and there is an ongoing work to support AM API v3 [58].

AM API v2 Methods:

GetVersion
Syntax: struct GetVersion([optional: struct options]) **Functionality:** Get static version and configuration information about this aggregate. Return includes: · The version of the GENI Aggregate Manager API supported by this aggregate manager instance. · URLs for other versions of this API supported by this aggregate · The RSpec formats accepted at this aggregate · Other information about the configuration of this aggregate. **Parameters:** options: Optional **Returns:** a struct where the value member is Version Information

ListResources
Syntax: struct ListResources(string credentials[], struct options) **Functionality:** Returns a listing and description of available resources at this aggregate, or resources allocated to a named slice at this aggregate. The resource listing and description provides sufficient information for clients to select among available resources, or to use reserved resources. These listings are known as RSpecs. **Parameters:** credentials[]: An array of caller credentials options: Indicate the set of resources that the caller is interested in, or the format of the result **Returns:** a struct where the value member is an Advertisement RSpec

CreateSliver

Syntax:
struct CreateSliver(string slice_urn,
string credentials[],
string rspec,
struct users[],
struct options)

Functionality:
Allocates resources as described in a request RSpec argument to a slice with the named URN. This operation is expected to start the allocated resources asynchronously after the operation has successfully completed. Callers can check on the status of the resources using SliverStatus. Resources will be reserved until a particular time, set by the aggregate according to policy. That expiration time will be no later than the expiration time of the provided slice credential. This method returns a listing and description of the resources reserved for the slice by this operation, in the form of a manifest RSpec.

Parameters:
- slice_urn: The URN of the slice to which the resources specified in rspec will be allocated
- credentials: An array of caller credentials
- rspec: Request RSpec to allocate
- users: List of users
- options: Optional

Returns:
a struct where the value member is a Manifest RSpec

DeleteSliver

Syntax:
struct DeleteSliver(string slice_urn,
string credentials[],
struct options)

Functionality:
Deletes any slivers at the given aggregate belonging to the given slice, by stopping the resources if they are still running, and then deallocating the resources associated with the slice. When complete, this slice will own no resources on this aggregate - any such resources will have been stopped.

Parameters:

slice_urn: The URN of slice to deallocate from
credentials: List of credentials
options: Optional

Returns:
a struct where the value member is a Boolean

SliverStatus

Syntax:
struct SliverStatus(string slice_urn, string credentials[], struct options)

Functionality:
Get the status of a sliver or slivers belonging to the given slice at the given aggregate. Status may include other dynamic reservation or instantiation information as required by the resource type and aggregate. This method is used to provide updates on the state of the resources after the completion of CreateSliver, which began to asynchronously provision and start the resources.

Parameters:
slice_urn: The URN of slice
credentials: List of credentials
options: Optional

Returns:
a struct where the value member contains the status of the overall reservation

RenewSliver

Syntax:
struct RenewSliver(string slice_urn, string credentials[], string expiration_time, struct options)

Functionality:
Renews the resources in all slivers at this aggregate belonging to the given slice until the given time, extending the lifetime of the slice. Aggregates may limit how long reservations may be extended. Initial sliver expiration is set by aggregate policy, no later than the slice credential expiration time.

Parameters:

- slice_urn: The URN of slice

- credentials: List of credentials
- expiration_time: The date-time when the reservation should be extended until.
- options: Optional

Returns:
a struct where the value member is a Boolean

Registry API:

It might be helpful to stress that only the AM API is currently defined as part of GENI. In fact the original SFA document [1] also defined a PKI-like infrastructure for managing identities. Although the initial naming referred to this as Management Authorities, the name that is most often used these days to refer to this is the notion of Registry. A Registry is in essence a trusted body that issues certificates (or GIDs) to users or sub-authorities, and Registries are organized in a tree that implements the underlying hierarchical naming space, and play a fundamental role in the web of trust among the federation. So it is of course mandatory to run at least one such Registry in a production federation.

Further Specifications of SFAWraps SFA Registry Methods can be found in Appendix B: Further Specifications for SFAWrap Registry API Methods.

Description of AMSoil

Introduction

AMSoil is one of the available tools that can be used by testbeds in order to implement an SFA API.

In the OFELIA project [59] it was agreed to build the OpenFlow islands control framework by following the SFA [1] architecture. During the OFELIA Control Framework (OCF) development, the need to create a new and faster way to adapt the new resources to the OCF, and consequently to SFA was identified. In order to solve this they developed an AM base class named AMsoil [62].

AMsoil is a framework for building aggregate managers; the aim is to support AM developers in creating software for managing resources. AMs are the work-horse between the client (e.g. gcf's OMNI [29]) and the actual resource.

AMsoil contains a set of tools which AM developers need on a regular basis including:

- Configuration management
- API management (e.g. via XML-RPC)
- Context storage
- Policy management

- Asynchronous task management
- Internal notification system

It provides communication interfaces, management of bookings, configuration facilities, logging facilities and an authentication and authorization mechanism. The main goal is to decouple the implementation efforts on API changes, new resource types and base class improvements.

AMsoil encapsulates the core functionality of an Aggregate Manager (AM) and provides a resource-independent extension mechanism to implement AM. Due this decouple, it could be logically divided in two parts: a common part and a resource-specific part.

The base class shall serve as base for all AM implementation and it is structured to:

- Be a starting point for quickly developing an AM.
- Handle all operations which are resource independent.
- Evolve the common functionality without having to change the resource-specific management.
- Support different interfaces.
- Support different Authentication and Authorization-schemes.
- Be pluggable for other plug-ins such as policy management, logging, etc.
- Extendable for other functionalities without affecting the commonly developed AMsoil code.

AMsoil design:

As previously stated, there are common parts and resource specific parts within the AMsoil. Figure 15 shows how the AMsoil has been designed in order to obtain enough disengagement between these parts:

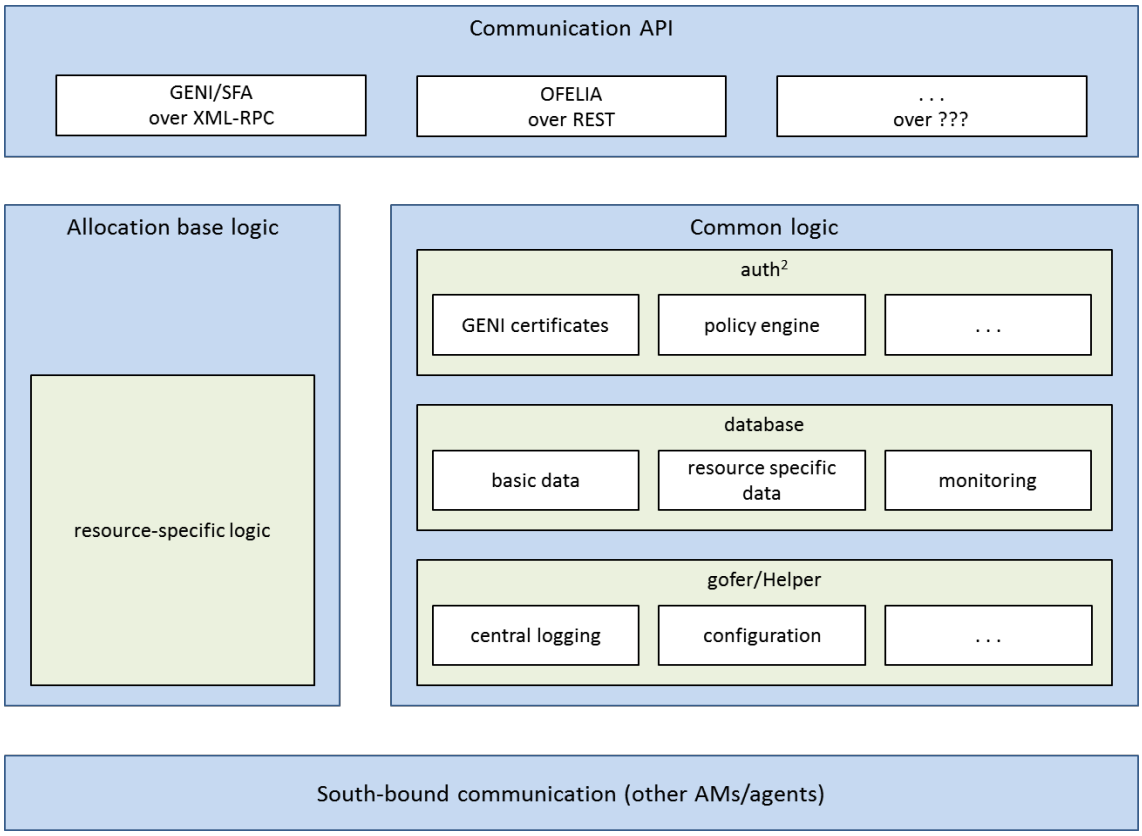

Figure 15: AMsoil Architecture

Figure 15 shows that north and south boundaries are the communication layers. The north one allows users (or other AMsoils) to communicate with AMsoil and its aim is to support different kinds of APIs. The same could be said about the south boundaries. Regarding the common part, we can see in the architecture design that it supports more features than only some resource creation. The objective is to have management of reservations, configuration facilities, logging facilities and an authentication and authorization mechanism. These features are plugin-based. A plugin provides a feature to the AM. If a feature is already provided by the testbed, then the plugin is not needed.
For further understanding, we will explain the typical workflow for a resource creation request:

1 An RPC (Remote Procedure Call) receives the request and processes the input and looks up Adapters in the AdapterRegistry.

2 The AdapterRegistry returns a list of Adapters which support certain contracts regarding RPC calls.

3 The Adapter then translates between the resource management and the RPC calls (e.g. translating to XML). The AdapterRegistry is the only instance which knows about the concrete shape of a Resource and the formats a RPC requires.

4 When adapting the RPC request, the Adapter asks the ResourceManagerRegistry for ResourceManagers which support the resource type which is asked for.

5 The ResourceManager is a "database" for Resources so you can call, find and reserve on it.

6 Finally, the Resource is concerned about the concrete handling of resources (e.g. starting, keeping allocation times).

This is depicted in Figure 16:

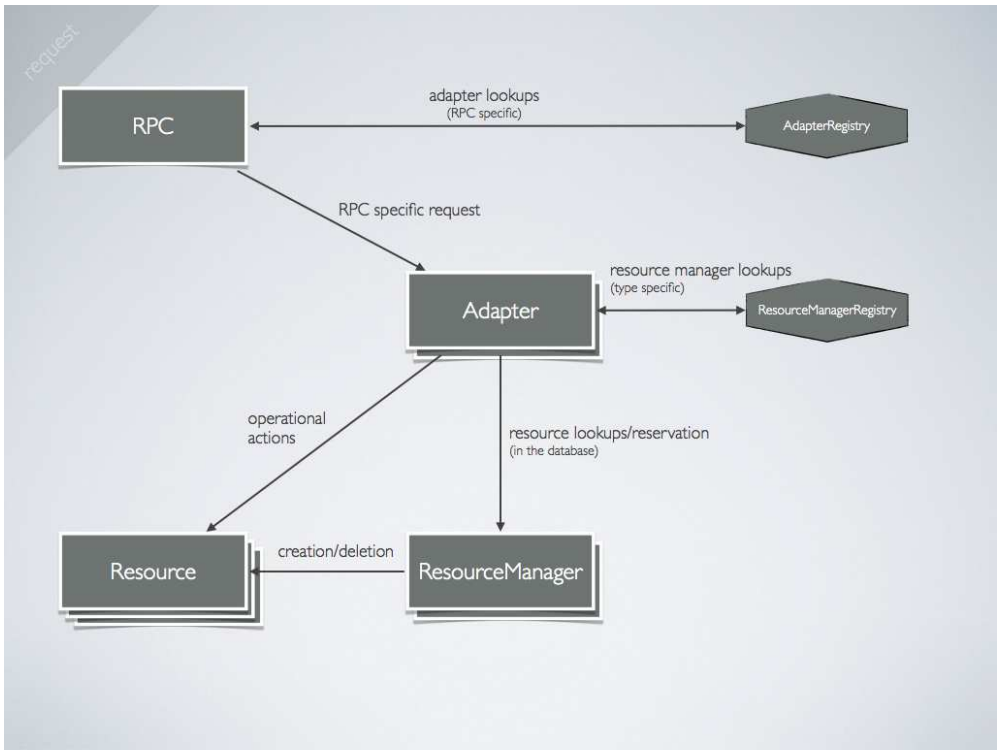

Figure 16: Typical AMsoil workflow for resource provisioning

The idea behind this architecture is that the RPC shall only deal with communication layer matters and the ResourceManager should offer a decent API which ensembles requirements of this concrete resource type. The actual Resource then should know how the concrete resource needs to be handled. And finally to have ResourceManagers and RPCs work together we need a translator: the Adapter.
Furthermore, the component of this architecture will be explained grouped in common and specific part.

The common part

It manages tasks which are needed by each AM, such as identification, authentication and authorization, interface compliance, communication with resources and its managers.

- The Adapter then translates between the resource management and the (e.g. translating to XML). The AdapterRegistry is the only instance aware of the concrete shape of a Resource and the formats a RPC wants.
- When adapting the RPC request, the Adapter asks the ResourceManagerRegistry for ResourceManagers which support the resource type it is asked for.
- The ResourceManager is a "database" for Resources, so you can call, find, and reserve on it.

The resource-specific part

It implements the actual handling of each resource, e.g. talking to an agent and allocating resources. AMsoil is used as a container which incorporates the VM Manager functionality as a resource-specific part. This way, a more standard OCF VM Manager will be achieved.

- An RPC receives the request and processes the input and looks up Adapters in the AdapterRegistry.
- The AdapterRegistry can return a list of Adapters which support certain contracts regarding RPC calls.
- Finally, the Resource is concerned about the concrete handling of resources (e.g. starting, keeping allocation times).

Additional information is provided in [62].

Development status

AMSoil is currently under development, and right now only the core functionalities are implemented, also because of the structure based on plugins the API of the AMsoil tool is always dependent on which tool is used or implemented.
Currently AMsoil supports GENI AM API, version 2. By separating the actual resource management logic and the API, AMsoil decouples these two interfaces. This makes it easy to implement additional APIs later on and enforces re-usable code for resource management and API handling

3.6.5 Specifications for AMSoil

AMSoil Specifications and Examples can be found in Appendix C: Further AMSoil specifications and examples

3.6.6 Required additional implementations

Functionality	Cycle 1	Further Cycles
Support of GENI AM API v3	• Add the support for the GENI Aggregate Manager API v3.	
Import Trusted Roots Certificates	• WP7:"Trustworthiness" will provide the federation with a central server containing all the trusted roots certificates that SFAWrap will need to be able to import.	

3.6.7 Requirements for testbeds to adopt SFAWrap

Testbeds aiming to be wrapped with SFAWrap in order to be SFA compliant need to:

- Implement their testbed driver for SFAWrap by overriding the driver class of SFAWrap.
- In case the testbed manages a user base, the list of users with their respective SSH Public Keys and associated slices, will need to be imported to the SFAWrap Registry.
- In case the testbed is exposing specific resources that are not supported by the current SFAWrap RSpecs, the testbed will need to implement its own version of RSpecs.

3.6.8 Requirements for testbeds to adopt AMsoil

There are no specific requirements for testbeds aiming to adopt AMsoil in order to be exposed through SFA. Besides having an Aggregate Manager-like structure or similar in which each component can be encapsulated inside the AMsoil base-class, they should be testbeds which also require the other functionalities provided by AMsoil. If the testbed already have those functionalities, they can be replaced by those provided by AMsoil, that is, replacing the testbed's current management software. Or, in an easier option, adopt SFAWrap. I.e. the component for managing a kind of resource is fit into the base-class getting their calls translated to SFA and also gaining the other functionalities provided by AMsoil.

As for the technical requirements, they can be summarized as:

- Have a Python v.2.7 or higher version installed
- Install the plugin dependencies defined in the repository webpage [3]
- Follow the instructions described in the previous section or in the code repository.

3.7 Experiment control

3.7.1 General description

In the context of experiment control in Fed4FIRE, two main aspects can be identified. The first one is the need for a federated resource control layer. The second one is the need for an experiment controller which makes use of this control layer to actually control the experiment. Both aspects are introduced in this section in more detail.

3.7.1.1 Federated resource control layer

A desired outcome of Fed4FIRE is to enable experimenters to easily access and control resources offered by a large range of facilities using a single experiment control tool. However, facilities have different management software, and provide different interfaces and ways of accessing and controlling resources. In order to make it possible for a single experiment control tool to control resources on facilities using different management software, some form of resource control federation is needed across the facilities. This is to say that, for it to be possible for a number of experiment control tools to operate on a large number of facilities at a reasonable cost, without having to implement specific code to interact with each different facility, a common interface or protocol to interact with resources must exist in all facilities.

Before we can start defining and implementing higher level features for experiment control, and test them over a large range of facilities, this federated resource control layer must be defined, since it provides the base building block for experiment control federation.

There are many ways in which such resource control federation can be achieved; a simplistic one would be to impose on testbed owners to use the same management software for all testbeds. This is however not an option since a significant effort has been invested in developing such software and is it not feasible nor desirable to produce a management software that would satisfy the requirements of all facilities within the duration and budget of the project.
A better alternative is to support a common resource control interface in all facilities which can co-exist with existing management software. This alternative is in line with the choice of a heterogeneous federation architecture taken in D2.1, where it says “Heterogeneous federation:
in this architecture, each testbed keeps its own management software, but interfaces on top of the testbed software are specified, standardized and made interoperable to a federation.”

A standardized resource control protocol will permit to control resources provided by federated facilities using different management software in a uniform way.
The novel federated resource control protocol (FRCP) is such a protocol. It consists of a message being sent by a requester to a component (or resource). The component may accept the message and perform the requested associated actions. For the message exchange with the resources (physical or application resources), the necessary resource controller (RC) implementation, supporting the set of defined messages, should be running in the different resources of the facility. However, for cycle 1 there is a lot of work already done by NICTA, defining the messages characteristics for the FRCP, and several resource controllers suitable for a wide range of testbed, in the context of OMF 6 development.

During the first cycle, taking into account that defining the protocol specification can take several years, as it was the case for SFA, our efforts will be addressed in analysing and testing the NICTA implementation of the FRCP. But at the same time, different partners in this project such as INRIA and TUB, already started the discussions that will lead to standardization. The current description of the protocol can be found in the appendix.

Existing testbed management software should implement an interface compatible with the federated resource control protocol. A reference implementation of such interface will be provided in this document, based on the previous experience in supporting the RC for virtual machines in PlanetLab testbeds.

Testbed owners can opt to adopt the reference interface, or alternatively to implement a new one if this doesn't suit the needs of their facility. However it is not expected that all available resources on all testbeds will be controlled from a single experiment control engine at the end of cycle 1.

Experiment controller engines compatible with FRCP will be able to access and control resources provided by federated facilities out of the box. The following Figure 17 shows FRCP in the context of WP5 components.

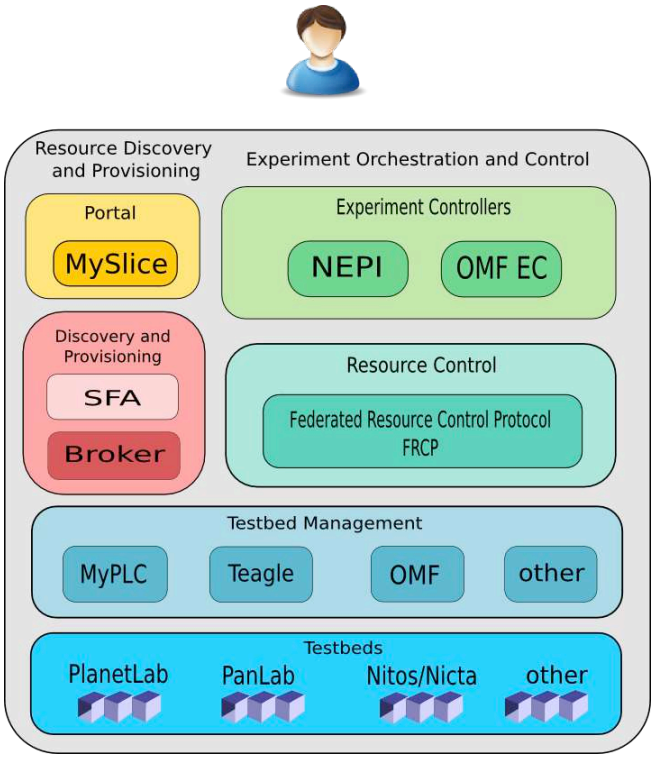

Figure 17: FRCP in the context of Experiment Lifecycle Components

Some main considerations and requirements for the FRCP implementation identified in WP5 are the following:

- Testbeds should not be required to change their management software and extra software should be kept to a minimum
- It should be possible to control any type of resources

- The current FRCP defines a mechanism for signing each message to securely bind the message to a 'sender'. While it is outside the scope of the FRCP specification, the recommended 'best practice' is to use PKI public/private key encryption with X509 credentials. Further details of FRCP authentication and authorization have to be worked out in collaboration with WP7.
- Centralized infrastructure should be kept to a minimum
- Each facility should remain to be independent and not depend on any new infrastructure
- FRCP is not management software, it is not supposed to re-implement management software functionalities, only provide an interface to control resources
- FRCP implementation should be kept as modular as possible, so any modules can be adapted independently to the requirements of the facility

3.7.1.2 Experiment control engine

The experiment control engine or experiment controller (EC) is a user tool where the user can describe experiments, and the EC is responsible for the experiment orchestration. The EC can benefit from existing tools to solve different stages from the experiment life cycle, for example in can benefit from a SFA frontend to discover and provision resources from a testbed facility. It can also query a central measurement repository to retrieve results from the experiment. The idea of the EC is not to implement every step necessary to deploy an experiment, but to support the required framework.

To be able to control resources in a uniform way, any candidate experiment control engine must support the chosen FRCP. Additionally, the following list of features should be taken into account:

1) Description language. The EC must provide an experiment description language which allows running experiments programmatically through scripts. The ability to alternatively describe experiments graphically is also desirable.
2) Authentication, authorization, access policies. This feature depends on WP7. Long running experiments should be able to be controlled by more than one person/agent. Each FRCP message is associated with a single entity (the one signing the message), but it's up to the policy of the receiver to decide to act or not.
3) Resource description and discovery. This feature depends on T5.2. Resource description, which should initially be done using RSpecs and discovery through SFA or a SFA proxy such as MySlice.
4) Resource reservation. This feature depends on T5.3. Resource reservation can be achieved through SFA, since SFA is able to handle both, shared and exclusive resources. SFA through the control and management software from the testbed will interact with the corresponding scheduler.
5) Resource access. This feature refers to, once the resource is provisioned by SFA, the mechanism to provide the experiment controller with the information needed to access the resource (i.e. to which service to talk to, to control the resource)

6) Resource bootstrapping (e.g. copy initial image on node). Further discussions are needed in order to be able to define if this feature will need to be performed by the experiment controller or is the testbed and the provisioning system that will take care of it.
7) Time base and state based task execution. This feature will be clearer once native FRCP task scheduling capabilities are defined.
8) Measurements. Interfaces and mechanisms in which monitoring data will be available for the experiment controller are being defined in WP6. *Hence this feature depends on WP6.*

The experiment controller during the first cycle will focus on addressing items 1 and 7. This means that, to execute tasks defined by the user and to run applications, we will assume that at the moment that the experiment controllers comes into play, that all the necessary previous steps of the experiment lifecycle have already been executed using other functional elements of the Fed4FIRE architecture. In other words, from the experiment controller viewpoint it is expected in the first cycle that resources will be already provisioned and ready to be accessed and controlled at the moment that the experiment controller is activated.

In the context of the next cycles, it is important to mention that (as set out in item 6) there is a diffuse boundary defining who is in charge of setting up resources for the experiment. The FRCP could be in charge of resources related to services and applications, but it seems more logical that the testbed facilities as part of the provisioning process, involving their control and management software, install the necessary RC, and guaranty the access once the resource must be provisioned, to be able to accede and control them. This needs further discussion and clarification.

3.7.2 Evaluation of possible approaches for implementation

As introduced in the previous section, the proposed architecture for experiment control federation has two main components:

1) the federated resource control protocol (FRCP)
2) the experiment control engine (or experiment controller)

This section will evaluate the different possible implementation approaches for both of them.

Evaluation of the Federated Resource Control Protocol (FRCP)

For the federated resource control protocol five approaches have been analyzed

a) to adopt the FRCP protocol defined in the Architectural Foundation For Federated Experimental Facilities (AFFEF) proposed by NICTA [86],
b) to adopt FITeagle T1 interface [87],
c) to adopt ORCA [89],
d) to adopt BonFire architecture [88],
e) to create a new solution from scratch.

Approach a) is based on the adoption of AFFEF, and the FRCP protocol that is part of it. This FRCP protocol may be used by any software components for controlling and orchestrating distributed

resources, such as testbed devices, sensor nodes or measurement software. This protocol is currently used by OMF6 entities, and may be implemented by any other software project to interact with other FRCP-enabled components. The basic protocol consists of a message being sent by a requester to a component (or resource). The component may accept the message and perform the requested associated actions. This may lead to further messages being sent to an observer. This separate observer is introduced to allow for different messaging frameworks. The protocol consists of five messages: inform, configure, request, create, and release. The FRCP messages can be described using either an XML or JSON format. The main advantage of this approach is that NICTA is already working to adapt the OMF testbed management and control framework to support **AFFEF (and hence the corresponding FRCP layer) in OMF version 6**. OMF is already in use by several testbeds participating in FED4FIRE (e.g. NORBIT [39], NITOS [51], w-ilab.t [36], NETMODE [38]) and has shown to be interoperable with other testbed management software (e.g. MyPLC [50] used in PlanteLab Europe [49]). Furthermore, the AFFEF proposal satisfies cycle 1 requirements for experiment control.

Approach b) is based on the **FITeagle T1 interface**. FITeagle [6] is the central coordination instance that holds together all PanLab test labs providing maximum range of testing and prototyping possibilities. Through FITeagle [6], it is possible to browse resources provided by Panlab Partner labs, configure, deploy, and register new resources to be provided by the federation.
FITeagle offers the Virtual Customer Testbed (VCT) Tool [87] and a Portal for user interface, it incorporates a repository implementation, allows search and configuration of testbed components, allows the remote provisioning of testbed resources and scheduling for booking resources. The PanLab testbed resources can be physical machines, virtual machines, different types of software and devices.
The architecture consists in different components such as the Domain Managers, for controlling resources inside a domain, the Resource Adaptors used as devices drivers to translate federation level management commands to resource specific communication (e.g. SNMP, CLI, proprietary) and the T1 Teagle interface which instructs domain managers via a common control framework, and user interfaces for infrastructure design, configuration, and setup.

ORCA [89] lies at the base of approach c). ORCA is a software framework and open-source platform for managing a programmatically controllable shared substrate, which may contain any combination of servers, storage, networks, or other components. This class of systems is often called cloud computing or utility computing.
The ORCA software is deployed as a control framework for a prototype GENI facility. GENI can be seen as an ambitious futuristic vision of cloud networks as a platform for research in network science and engineering.
An ORCA deployment is a dynamic collection of interacting control servers (*actors*) that work together to provision and configure resources for each guest according to the policies of the participants. The actors represent various stakeholders in the shared infrastructure: substrate providers, resource consumers (e.g., GENI experimenters), and brokering intermediaries that coordinate and federate substrate providers and offer their resources to a set of consumers.

ORCA is based on the foundational abstraction of resource leasing. A lease is a contract involving a resource consumer, a resource provider, and one more brokering intermediaries. Each actor may manage large numbers of independent leases involving different participants.

Approach d) adopts the **BonFIRE architecture** [88]. BonFIRE is an EU project which is designing, building and operating a multi-site cloud-based facility on top of six infrastructure testbeds operated by six project partners. The infrastructure sites offer heterogeneous Cloud resources, including compute, storage and network.
The BonFIRE architecture is designed to support research on applications, services and systems targeting in particular, but not exclusively, the Internet of Services (IoS) community. It has key functionalities such as monitoring at infrastructure and virtual machine level, experiment management with a single declarative experiment descriptor, elasticity, and resource management for deployment of application software over a variety of differently configured resources (compute, storage, and network).
BonFIRE is geared towards experimentation and research into Cloud/IoS, and offers the facilities to easily create, manage and monitor experiments, whilst giving the experimenters more information and control of Cloud resources than what is offered by other public Cloud providers.
Interactions with BonFIRE are done via a RESTful interface based on the Open Cloud Computing Interface (OCCI) [90], referred to as the BonFIRE API. Each of the infrastructure sites in BonFIRE are accessed through this API, which makes it very easy to conduct multi-site, geographically distributed, experiments. Moreover, all resources on all infrastructure sites are accessed with a single BonFIRE sign-on that you set up when you register an account.

Finally, approach e) assumes that the federated resource control protocol adopted by Fed4FIRE is a **clean slate design**, where the entire protocol is designed and implemented from scratch during the course of the project.

These five approaches have been carefully compared with each other. This analysis is presented in Table 19. **The outcome of this comparison is that in Fed4FIRE, the FRCP protocol that is part of AFFEF, and hence of OMF6, will be adopted as the federated resource control protocol.**

Approach	Advantages	Disadvantages	Selected as final approach
Adopt OMF 6 (AFFEF) [85], [86]	• Large active community of developers and users • Support for basic resources such as nodes, VMs, interfaces, applications, OpenFlow switches. • Straightforward development of additional new Resource Proxy	• Interface with SFA for resource discovery, allocation and provisioning is currently under development • Testbeds not using OMF as their management software will need to either adopt OMF or add	YES

	modules for other type of resources • Scale to hundreds of resources • Open Resource Model and Communication Protocol, allowing third party implementation of compatible resource proxies or experiment controllers • Can coexist with other testbed management software • Natively supported in FED4FIRE OMF testbed including Norbit, Nitos, w-ilab.t and Netmode. Additionally, compatibility of OMF with PLE and GENI testbeds has been shown.	extra interfaces to make their software compliant with the AFFEF	
Adopt Teagle T1 interface [87]	• Already a working federation • Provides abstractions to wrap resources and testbeds • Native support for FED4FIRE testbed (FuSeCo)	• An interface re-work is been planned and there is the intention to possibly adopt FRCP • Centralized portal type access • No information of scalability tests	
Adopt ORCA [89]	• Part of the GENI project which has wide spread in United States	• Federation only covers resource discovery and provisioning through distributed negotiation between testbeds • No real resource control interface • Does not support SFA	
Adopt BonFIRE [88]	• Already working federation • Supports multiple clients tools • Native supports for three FED4FIRE testbeds (Grid 5000, Virtual Wall, EPPC)	• Mostly cloud oriented (OCCI interface) • Assumed more or less uniform types of resources (VMs) • Centralized portal type	

		access • Resource configuration and control capabilities are not that flexible (mostly only during set-up with an init script)	
Clean slate implementation	• Clean design • Easy to extend in the future • OMF (or AFFEF compatibility) is not imposed as management software for testbeds	• Total amount of needed implementation effort exceeds the available manpower	

Table 19: Comparison of potential candidates for realizing Experiment Control in Fed4FIRE

Evaluation of the Experiment Controller

As presented in Table 20, three approaches have been analysed for realizing the experiment control engine (or experiment controller)

a) to use OMF EC,
b) to use NEPI experiment control framework and
c) to create a new experiment control engine from scratch.

From the three approaches, both approaches a) and b) have strong advantages. The OMF EC offers native support for OMF testbed management software, which is in use by many Fed4FIRE testbeds, and was selected as the base implementation for FRCP. However, adopting NEPI as an alternative experiment control engine is also desirable since NEPI allows to access and control potentially any type of resource, including emulated and simulated resources (e.g. ns-3 simulated nodes). Furthermore, NEPI supports resources discovery and provisioning through SFA, supports a graphical representation of the experiment model, supports interactive experiment configuration, and supports execution in background mode. More details regarding both these experiment controllers are given in section 3.7.3.

It is important to note that the choice of these two experiment controllers does not impair the existence of other experiment controllers that might wish to provide support for FRCP and thus work with the federation.

Based on this analysis, it is concluded that both OMF EC and NEPI are suitable tools as experiment controllers. Hence they will both be adopted as Fed4FIRE experiment control tools. The choice of using the one or the other will be related to the type of experiment.

Approach	Advantages	Disadvantages	Selected as final approach
Use OMF EC [91]	• High-level domain-specific language to describe experiments and automatically execute them through an experiment controller • Large user community already familiar with OEDL (OMF experiment description language) • Native support for OMF management software	• At the moment, it can't easily support arbitrary resources not running an OMF RC (e.g. ns-3 simulated nodes)	YES
Use NEPI [8]	• Supports OMF 5.4 • Support resource discovery and provisioning through SFA • Can potentially support arbitrary resources (including simulated and emulated resources) • Graphical experiment representation • Interactive configuration • Runs in background mode	• Doesn't support all OMF features as the OMF EC does	YES
Clean slate implementation	• Possibility of providing a new design that is better suited for FED4FIRE specific requirements	• Total amount of needed implementation effort exceeds the available manpower	

Table 20: Comparison of potential approaches for implementing Fed4FIRE's experiment control

3.7.3 Description of selected tools

In the previous section, an analysis was performed and a set of existing tools was chosen as the starting point for the implementation of both the Federated Resource Control Protocol and of the Experiment Controller. In this section, more profound details are given regarding these tools, and which of the needed functionality is already provided by them.

OMF Description

Overview

OMF is a generic framework which allows the definition and orchestration of experiments using shared (already provisioned) resources from different federated testbeds. OMF was originally developed for single testbed deployments, but has recently been extended to support multiple deployments and the following features. First, it provides a domain-specific language based on an event-based execution model to fully describe even complex experiments, such as ones involving cars or phones moving out of range of all control networks. OMF also defines a generic resource model and concise interaction protocol, which allows third parties to contribute new resources as well as develop new tools and mechanisms to control an experiment. It has a distributed communication infrastructure supporting the scalable orchestration of thousands of distributed and potentially disconnected resources.

In addition to these recent features, OMF also easily integrates with systems from other complementary testbed frameworks. Indeed, it interfaces with SFA (work-in-progress) to allow researchers to discover and provision resources to be used in their experiments. It is also compatible with measurement resources developed using the OML instrumentation and monitoring system. Finally it can also be paired with a web-based interface to act as a digital laboratory notebook and capture study cycles involving multiple experiments, through the use of additional wiki, versioning and statistical analysis capabilities.

Architecture

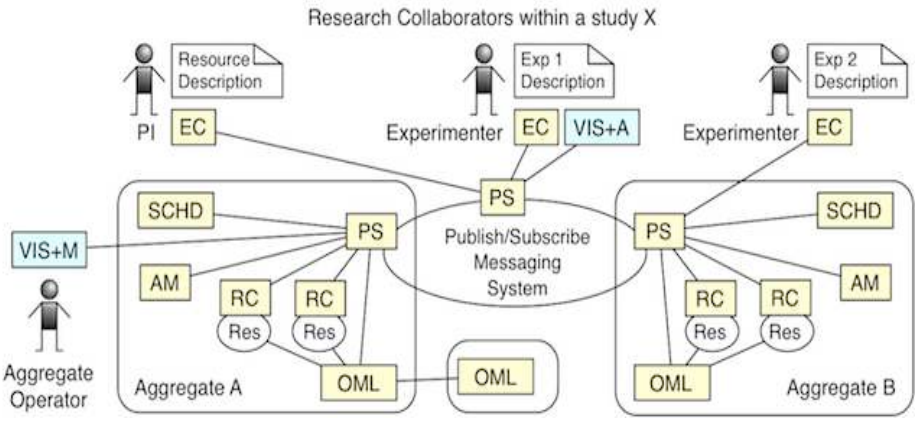

Figure 18: OMF Architecture

The above figure presents an overview of OMF architecture. At the centre of this architecture is a distributed publish-and- subscribe messaging system (pub/sub), which is realised by a set of Peering Servers (PS). OMF use a topic-based messaging pattern where resources or groups of resources are represented by topics. Communication among all entities is achieved by publishing and subscribing to the respective topics.

Any resource is associated with a Resource Controller (RC); the RC runs on the resource, a resource controller for wireless node is installed in a wireless node allowing control over it. For applications running in this node, a resource control of the kind application, will allow execution and control, start, stop, etc, over the application running. The RC is the proxy for one or more bare-bone resources. The RC subscribes to the topics associated with its resources and translates messages it receives from other entities to resource specific interactions. The RC normally also includes a policy component which first checks the validity of an incoming request.
An experimenter describes her experiment using a domain-specific language, and passes this description to an Experiment Controller (EC). This EC interprets the experiment description and uses the pub/sub system to send requests to the involved RCs and to receive reports from them on the experiment's progress. These RCs instruct the resources to execute the tasks within these requests and relays their outcomes. If a resource is instrumented, it may collect filtered measurements as instructed by the RC based on the experiment description. These data are sent to measurement entities using the OML Measurement Library, which can process them and store or forward them to other OML components. These OML entities are themselves resources, which understand the same communication protocol as the RCs and thus can also be organised and controlled via the EC and the experiment description. The experimenter may use additional software (Vis & A) to visualise the experiment's progress and analyse its collected measurements. More information about this is expected from WP6.

Resource Model and Life-Cycle

Our first design decision is to consider every entity participating in an experiment as a resource, independent on who is providing it. A resource has a set of properties and an associated life-cycle. It communicates with other resources through well-defined messages. It will be part of the first cycle to consolidate and justify that this approach is appropriate.

A new resource is created by an existing resource receiving a create message. It is initially in the inactive state, and may transition to the active state either immediately or at some later stage.
Given the large variety of resources there is no support for 'action' commands such as 'start' or 'doX'. Instead, resources are requested, through a configure message, to adjust their internal operations, so that the observable properties reach a certain value. In other words, we request the outcome and leave the 'how' to the resource. Some resources may only accept configuration requests in the inactive state, and some properties may only be set at creation time, as part of the create message. The request message asks a component to report on its status via an inform message. Finally, a component is discarded when it receives a release message.
Creating new resources by sending it to an existing one establishes a clear policy context in which we can decide if the request is valid or not. This results in a parent-child between the creator and the created. As a consequence and to maintain a proper accountability chain, a parent can only cease to exist when all its descendants have been released as well. In our current implementation, every resource maintains a list of its children and when receiving a release message it will forward it to its children as well. The initial release request will only succeed if all descendants have released themselves.
To support scalability we also introduce a group resource which maintains a set of other resources. In practice, this type of resource is simply a group messaging mechanism represented

by a pub/sub topic. All members of a group are requested to also subscribe to the respective group topic and process the received messages accordingly. Given the one-way communication pattern of pub/sub there is no additional semantics associated with group resources. For instance, there is no implied guarantee that a message sent to a group will be received by all its members. In fact, that guarantee does not even exist for individual resources. The life-cycle model can be extended with sub-states and transitions for any given type of resources. For example, a mobile robot resource might have the sub-states active/moving-forward or active/rotating.

Publish and Subscribe Messaging

We implemented the OMF pub/sub messaging system using the XMPP protocol and existing XMPP servers such as OpenFire. This system is composed of distributed servers peering with each other, and hosting topics. Authenticated clients can connect to a server, subscribe to any topics hosted by any servers and publish messages to them. XMPP's server-to-server protocol ensures that a message published to a topic is forwarded to all of its subscribers regardless of which server they are connected to.

Authentication and Authorisation

OMF needs to guarantee the publisher's identity for all messages. It uses end-to-end authentication based on the well-established practice of digital signature backed by a PKI infrastructure. Each OMF entity has a set of public and private keys, signs its generated messages with its private key, and verifies the signatures of received messages with the originator's public key. Public keys of a slice's entities are exchanged at resource provisioning between the users and the resources, or obtained via a trusted scheme such as a certificate authority or web of trust. The former method is currently implemented in deployed OMF testbeds. The other method is under evaluation as it may not scale to a high experiment churn involving large resource numbers. PlanetLab uses a similar key-based authentication scheme, which allowed OMF RCs to be deployed on its slivers.

An OMF entity also needs to verify that the originator of a message has sufficient rights for any enclosed requests. For example, although a user acquired the right to use a spectrum analyser to collect some data, it may be limited to use only some frequency ranges. We propose to attach a set of assertions or their resolvable references to each message, which establish the originator's rights. In any case, this proposal needs to be examined in the light of federation wide authorisation to determine whether it is suitable.

In the previous example, the user's request will have a first assertion from her institution confirming her affiliation to it, then a second assertion from the owner of the resource giving rights on a set of ranges to affiliates of that institution for a reserved time period. All assertions are signed by their originators and verified using the same key mechanism as above. Some assertions such as the last one in this example may be generated during the resource reservation and provisioning phase. This scheme can be considered a restricted, but light-weight variant of ABAC. It is restricted as it does not allow for additional rules to be attached to assertions and it is light-weight as most assertions will only be passed by reference and is based on widely adopted industry standards.

Experiment and Resource Controllers

As previously mentioned, the EC is the entity responsible for orchestrating the experiment. It interprets an experiment description from the user, developed using a domain-specific language, and sends related commands to RCs using the pub/sub communication scheme and our defined resource protocol.

The RC was designed to implement the resource model mentioned earlier. This generalises its source code to support different types of resources, such as virtual machines, mobile phone applications, or wireless sensors in a consistent manner. It also provides a reference implementation of our resource model and protocol, which serves as a base for custom RC implementations by third parties.

OMF experiments are fully event-driven, i.e. the experimenters define events associated with tasks in their experiment. These events are synchronisation barriers based on either time or values of properties or measurements from resources, when their conditions are met the tasks associated to them are executed. An example event could be “when all traffic generator have sent 10Mps of data” and the associated tasks could be “pause them, decrease rate by X, and resume them”.

The communication stack of all OMF entities supports the above messaging system and structure. This allows the EC and RCs to be on different network domains, removes the need for a permanent connection between them, and provides scalable group communication. It also enables new experiment capabilities, such as disconnected experiments (e.g. a mobile resource temporarily out of range of an EC), or long-running surveys (e.g. EC connecting episodically to RCs in a x-month data collection). Finally, the EC and RC entities also support the authentication scheme described above, which allowed RCs to be readily deployed on PlanetLab, and enabled orchestration of federated experiments.

Interface with Resource Discovery and Provisioning, and other tools

The use of a distributed pub/sub scheme as the core communication system of OMF allows it to easily interface with existing resource discovery and provisioning solutions. Indeed, the scheduler, registry and aggregate manager functionalities often defined in contributions from the GENI or FIRE initiatives can all be adapted to exchange messages using the OMF pub/sub system.

However to make the pub/sub system a central architectural component for Fed4FIRE, we need to carefully evaluate performance and impact in the communication process.

The challenge remains then in the sequence of interactions between these entities and the OMF-F ones in order to provide a seamless transition between the discovery and provisioning phase to the experiment orchestration phase. There is an ongoing collaboration with the developers of the NITOS Scheduler and the PlanetLab Europe SFA to address these challenges. As an example of this work in progress, we are developing an SFA-compliant OMF module, which should be released soon.

Similarly, this core communication system allowed the interface of OMF with a web-based digital laboratory notebook, which provides integrated functionalities allowing the record of research notes and experiment designs, the versioning of experiment descriptions and associated software, their automatic orchestration, and the processing of the result with a powerful analytical and statistical tool.

NEPI description

Overview

NEPI, the Network Experimentation Programming Interface, is a life-cycle management tool for network experiments. The idea behind NEPI is to provide a single tool to design, deploy, and control network experiments, and gather the experiment results. Going further, NEPI was specially conceived to function with arbitrary experimentation platforms, so researchers could use a single tool to work with network simulators, emulators, or physical testbeds, or even a mixture of them.

NEPI supports conducting hybrid-experiments, by deploying overlay topologies across many testbeds and communicating them through special tunnelling components.

To accomplish this, NEPI provides a high-level interface to describe experiments that is independent from any experimentation platform, but is able to capture platform specific configurations. Experiment definitions can be stored in XML format to be later reproduced, and modified according to experimentation needs.

Experiment execution is orchestrated by a global experiment controller, that is platform independent, and different platform-dependent testbed controllers, creating a control hierarchy that is able to adapt to platform specific requirements while providing an integrated control scheme.

Experiment life-cycle support

NEPI is a life-cycle management tool for network experiments. It covers the design, deployment, control and result gathering stages of the network experiment life-cycle as defined within the NEPI context:

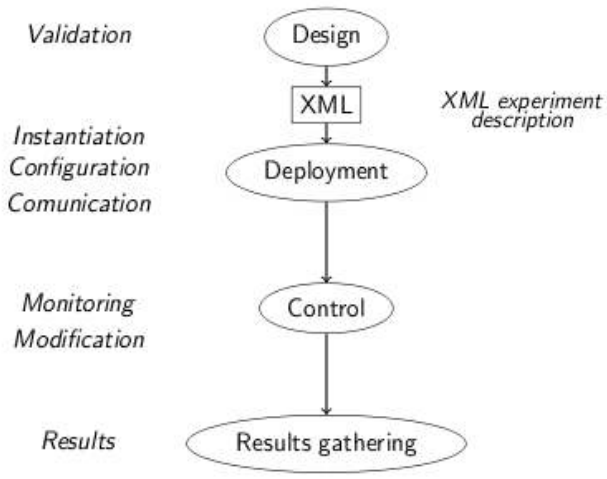

Figure 19: NEPI - Experiment Lifecycle Management

Along with what was established in section 3.7.1, during the first cycle, NEPI will focus on supporting design, and control through the federated resource control protocol. To support the whole experiment life cycle, the appropriate course of actions will be defined after development

cycle one, when authentication, authorization, discovery, provisioning, reservation and measurements according with other WPs are in more mature state.

Design

During the design stage the user constructs the experiment description using interconnected *box components*. A box component is defined by a type and by the experimentation platform (or testbed) it is associated to. An example of a box component is a PlanetLab Node box. The user can alter the experiment configuration by setting values on the box components *attributes*. NEPI will automatically validate connections between boxes and boxes attributes. Box components have a list of *traces*, which represent result files that can be activated to be generated during experiment execution. An example of trace is a *tcpdump* on a Network Interface box. The experiment description can be persisted to *XML* format. This description will be the input to NEPI's Experiment Controller to perform the deployment of the experiment. NEPI also supports graphical design through its GUI, Network Experiment Frontend (NEF).

Deployment

During deployment, NEPI uses the information on the *XML* description generated during design to instantiate, configure, and connect experiment components (resources). An *ExperimentController* object is responsible for processing the *XML* description, instantiating a specific *TestbedController* object for each testbed instance present in the experiment description, and issuing the right commands to each of the *TestbedControllers* so they can create the necessary experiment components (Nodes, Interfaces, Tunnels, etc). For example, an *XML* describing a PlanetLab testbed instance will trigger the instantiation of a PlanetLab TestbedController. Then, a nested description of a PlanetLab Node box will make the *ExperimentController* object send a message to the *PlanetLab TestbedController* object to handle the 'creation' (i.e. provisioning) of a PlanetLab Node resource. In turn, the *PlanetLab TestbedController* will locate the specified node and add it to the user's PlanetLab slide. Similarly, an *XML* describing a ns-3 simulated Node connected to another Node through PointToPointNetDevices, will trigger the creation of a *ns-3 [93] TestbedController* instance the ns-3 C++ objects in s process running the ns-3 simulation. The experiment deployment consists of well-defined steps that resolve concrete operations. Globally, these steps are:

1) Testbed set-up and configuration
2) Component instantiation
3) Component configuration
4) Connection of components inside a testbed
5) Connection of components from different testbeds
6) Launch of applications

Control

The *Control* stage occurs after deployment, when the experiment is running. During this stage, the user can interact with the *ExperimentController* object and modify experiment parameters in real-time. The Experiment and Testbed controllers are able to execute in remote locations and communicate via special messages. NEPI's API also provides methods to obtain information on the state of the running applications or other components.

Results Gathering

Results can be retrieved, from any remote controller, in a centralized way, at any moment from the moment the experiment starts running.

Experiment design

NEPI uses a *Boxes and Connectors* modelling abstraction to construct the experiment design. Each supported experimentation platform defines a set of *box* types (identified by a FactoryId), which represents the conceptual constructive blocks of an experiment. These boxes can be associated through named ports called *connectors*. Each connector in a box has a different function. The experiment description is thus constructed out of a graph which has *Boxes* as vertex. The boxes present in the experiment description and the connections between those boxes will define the experiment topology, both at a physical (infrastructure) and application (services) levels.
Boxes also have a set of *attributes* that allow defining the experiment configuration, and *traces* that allow defining experiment results to be collected.

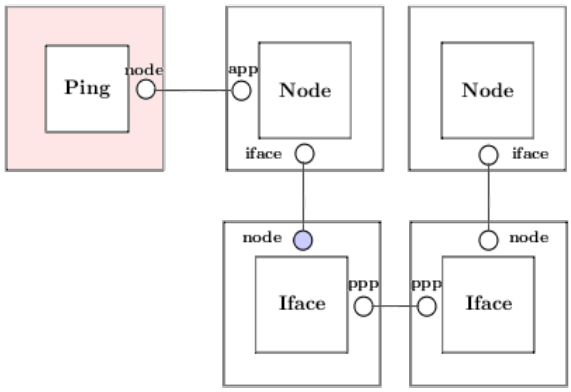

Figure 20: NEPI's Boxes and Connectors Modelling

All the information describing the *Boxes*, their Attributes, *Traces* and *Connectors* for each experimentation platform is defined in metadata files. At least one of those metadata files must exist for each supported platform. These metadata files also provide information to validate attribute values and allowed connections between Box connectors.
Object model

The main classes that participate in the experiment description are:

- **ExperimentDescription**: Groups the description of the different parts of the experiment that might be executed in different testbed instances.
- **TestbedDescription**: Describes the topology, applications, and configuration of the part of the experiment to be executed in a particular testbed instance.
- **FactoriesProvider**: Provides the box classes definitions for a concrete testbed type. (Ex: ns-3, PlanetLab)
- **Boxes**: Functional units that describe an experiment. (Ex: Node, Interface, Application, Channel)

Experiment script

NEPI provides an experiment description language in Python to programmatically describe and execute experiments. Additionally, a graphical user interface called Network Experimentation Frontend (NEF) [94] allows to graphically design and run experiments. The following is a step by step (Step 1…Step 8) example of a NEPI script of a simple Ping experiment using ns-3 simulator using NEPI Python libraries.

1. Import the necessary Python classes from NEPI design modules.

```
from nepi.core.design import ExperimentDescription,
FactoriesProvider
```

2. Instantiate the *ExperimentDescription* object.

```
exp_desc = ExperimentDescription()
```

3. Instantiate a *FactoryProvider* for a particular testbed type (Ex: ns-3).

```
testbed_id = "ns3"
ns3_provider = FactoriesProvider(testbed_id)
```

4. Instantiate a *TestbedDescription* and configure it.

```
ns3_desc = exp_desc.add_testbed_description(ns3_provider)
ns3_desc.set_attribute_value("homeDirectory",
"/tmp/experiment_home")
ns3_desc.set_attribute_value("SimulatorImplementationType",
"ns3::RealtimeSimulatorImpl")
ns3_desc.set_attribute_value("ChecksumEnabled", True)
```

5. Instantiate and connect some *Boxes*. (Ex: 2 ns-3 nodes connected though a point to point channel)

```
node1 = ns3_desc.create("ns3::Node")
ipv41 = ns3_desc.create("ns3::Ipv4L3Protocol")
arp1  = ns3_desc.create("ns3::ArpL3Protocol")
icmp1 = ns3_desc.create("ns3::Icmpv4L4Protocol")
node1.connector("protos").connect(ipv41.connector("node"))
node1.connector("protos").connect(arp1.connector("node"))
node1.connector("protos").connect(icmp1.connector("node"))
iface1 = ns3_desc.create("ns3::PointToPointNetDevice")
queue1 = ns3_desc.create("ns3::DropTailQueue")
node1.connector("devs").connect(iface1.connector("node"))
iface1.connector("queue").connect(queue1.connector("dev"))

node2 = ns3_desc.create("ns3::Node")
ipv42 = ns3_desc.create("ns3::Ipv4L3Protocol")
arp2  = ns3_desc.create("ns3::ArpL3Protocol")
icmp2 = ns3_desc.create("ns3::Icmpv4L4Protocol")
node2.connector("protos").connect(ipv42.connector("node"))
node2.connector("protos").connect(arp2.connector("node"))
node2.connector("protos").connect(icmp2.connector("node"))
iface2 = ns3_desc.create("ns3::PointToPointNetDevice")
queue2 = ns3_desc.create("ns3::DropTailQueue")
```

```
node2.connector("devs").connect(iface2.connector("node"))
iface2.connector("queue").connect(queue2.connector("dev"))

channel = ns3_desc.create("ns3::PointToPointChannel")
iface1.connector("chan").connect(channel.connector("dev2"))
iface2.connector("chan").connect(channel.connector("dev2"))
```

6. Set IP addresses on the network interfaces.

```
ip1 = iface1.add_address()
ip1.set_attribute_value("Address", "10.0.0.1")

ip2 = iface2.add_address()
ip2.set_attribute_value("Address", "10.0.0.2")
```

7. Enable *Trace* results.

```
iface2.enable_trace("P2PAsciiTrace")
```

8. Add and configure an application. (Ex: Ping)

```
app = ns3_desc.create("ns3::V4Ping")
app.set_attribute_value("Remote", "10.0.0.2")
app.set_attribute_value("StartTime", "0s")
app.set_attribute_value("StopTime", "2s")
app.connector("node").connect(node1.connector("apps"))
```

Experiment execution

NEPI uses a hierarchical control structure to orchestrate experiment execution. A single *ExperimentController* instance takes care of the global supervision of the experiment.

The user will communicate with this global controller, which in turn communicates with the testbed specific controllers, the *TestbedController* instances, which are in charge of supervising specific testbed instances.

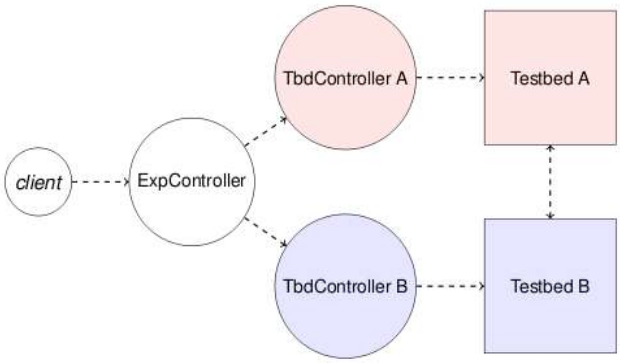

Figure 21: NEPI – Object Model

Object model

The main classes that participate in controlling the experiment execution are:

- **ExperimentController**: Global experiment orchestrator. It takes care of instantiating the different *TestbedController* instances and instructing them to create all the experiment components and connections.
- **TestbedController**: Testbed instance orchestrator. It takes care of performing testbed specific tasks, such as component creation and interconnection.

Experiment script

In order to deploy and control the experiment in NEPI it is be necessary to add specific lines of code to the previous design script.

1. Import the *ExperimentController* class

```
from nepi.core.execute import ExperimentController
```

2. Obtain the *XML* experiment description

```
xml = exp_desc.to_xml()
```

3. Instantiate an *ExperimentController* object, using the *XML* description, and start it

```
controller = ExperimentController(xml, "/tmp/experiment_dir")
controller.start()
```

4. Wait until some event occurs in the experiment. (Ex: The Ping application is finished)

```
while not controller.is_finished(app.guid):
time.sleep(0.5)
```

5. Get the trace result content

```
ping_result = controller.trace(iface2.guid, "P2PAsciiTrace")
print ping_result
```

NEPI and OMF EC can coexist, and will be chosen by the user, depending the need of each experiment.

3.7.4 Required additional implementations

As the first step to support the adopted federated resource control model and experiment controllers in cycle 1, new implementations and/or adaptations of them are needed. It is also possible that the testbeds should extend their management software with specific components. All these required additional implementations are described in this section.

NEPI support for MySlice (and the diversity of Rspec)

In the course of last year NEPI implemented an API to talk SFA directly, but with the appearance of MySlice, it was decided to rewrite this API to convert it into a MySlice Client in python to query MySlice API. The idea is to benefit from the filters and query convenience that MySlice API

provides, to solve the discovery, reservation and provisioning of resources, instead of parsing big Rspec.
Formerly NEPI had to query SFA deployments through the SFI client and parse the Rspec results. These Rspec are different from testbed to testbed with the complexity that this involves. Currently NEPI supports two Rspec definitions, PlanetLab Europe and NITOS Rspec.
With the development of NEPI's integration with MySlice a set of queries of interest had been tested, but the metadata available to ask to the different testbed resources, is associated to their description of resources in their corresponding Rspec. Therefore, as long as there are differences between how the same resources and properties are named, a post analysis of the query result must be implemented in NEPI, for every testbed available through SFA, which is the same as saying through MySlice. At the same time, MySlice is under development right now, so the NEPI-MySlice integration is in testing version, and therefore has to be supported completely with the release of MySlice.

NEPI support for OMF 6

NEPI is already supporting OMF 5.4 but needs to be extended to support the stable version of OMF 6. Presently NEPI is supporting communication with XMPP servers, and functionalities like create a topic, delete, subscribe, send message, etc. These can be reused for exchanging messages using the new control protocol.
NEPI development should focus on supporting the messages associated to the federated resource control protocol (FRCP), not only the creation of these messages, but also the processing and control model to support the architecture characteristics of the protocol, e.g.: how to model the different resources proxies and access them through a common resource controller.
Meanwhile provisioning is not handling the complete setup of the resources. Therefore NEPI will also need to support the image loading and saving for the nodes participating in the experiment, either using OMF commands or developing the corresponding method. The image will create resource proxies and controllers for the desired experiment.

OMF 6 stable version release

At the moment OMF 6 is in its beta release, but in order to be installed in testbeds it should be in its stable version. With the installation of the stable version, the testbed will provide the interface to interact with the federated resource control protocol in order to exchange the generic messages to control their resources. This is a strong requirement to be able to run the prototype demo experiment targeted in cycle 1.

OMF EC according to new architecture of OMF 6

With the release of OMF 6 and its new design and architecture, a new OMF experiment controller should be provided that supports it. For example, it should be able to create the new messages according to the FRCP. See API for more details in FRCP description, and what kind of messages the EC should support. Probably in the case of NEPI and OMF EC, as both support OMF 5.4, the new messages system will be an extension to the already supported one.

OMF6 compatibility in testbeds

Testbed owners need to provide OMF6 compatibility in their testbeds, either deploying OMF 6 as management software for the testbed, or deploying the corresponding interface (OMF 6 based) of the federated resource control protocol. Only some testbed owners, currently using OMF 5.4 as control and management software for their facilities already commit to deploy OMF 6 when stable.

Another important issue to address, is new developments of resource controllers. In case the ones existing in OMF 6 stable release do not cover all testbed resources functionalities, the testbed owner will have to provide the corresponding resource controllers. The new resource controllers can be adaptation and inherit from existing ones.

In summary, functionalities implemented by the end of cycle 1 regarding experiment control are listed in Table 21: Experiment Control functionalities to implemented for cycle 1

NEPI	Support for SFA frontend MySlice
NEPI	Support for OMF 6 and the FRCP
OMF 6	OMF 6 stable release, including OMF 6 EC and authorization support. Evaluation of the pub/sub messaging system
Fed4FIRE Testbeds	Installation of OMF 6 stable release, or testbeds' implementation of the FRCP (Resource Controllers and pub/sub system)
Fed4FIRE Testbeds	Implementations of the corresponding Resource Controllers for testbed specific resources

Table 21: Experiment Control functionalities to implemented for cycle 1

3.7.5 Specifications

A the moment, this section only covers the functional specification for experiment control since the technical specification for the federated resource control protocol and for the experiment control engine will be the same as those defined by AFFEF in OMF6. These specifications will only be finalized at the time that OMF6 is released as a stable version. This release is expected soon. Future cycles of Fed4FIRE however will attempt to refine these specifications and the developer documentation for the experiment controller.

Following the same convention as in the beginning of the experiment control section of this deliverable, the functional specification is divided into two parts, the necessary requirements to support federated resource control, and the necessary dependencies to support the different experiment control engines.

Federated resource control protocol

As it was already discussed, the benefit for testbed owners to adopt the federated resource control protocol lies in the ability to achieve federation for experiment control.
For the federated resource control protocol, as it is right now defined by NICTA in OMF, there has to be support for a pub/sub messaging system, in particular the testbed owner should provide an XMPP server. This XMPP server should be accessible from a private network, in order to connect with the testbed resources and resources controllers, but also it has to be connected to a public network, to allow the experiment controller to talk to the resources through it. The XMPP server could be installed in a DMZ to minimize security risks.
In previous implementations of OMF the same requirements apply for the aggregate manager, at present similar considerations for the broker should be taken into account.
The testbed should provide compatible disk images for their resources. In case the user does not need a particular image for his/her experiment, he/she should have the option to install a compatible OS.
Resources should be able to support PXE network booting, and for image loading to resources, the image server should be able to be contacted from the experiment controller.
In order to control testbed resources, the corresponding resource controllers should be implemented as part of the testbed deployment of the FRCP. This will be the responsibility of the testbed owner.

To illustrate the above specifications, previous efforts to introduce support for the OMF messaging system in the PlanetLab testbed are elaborated in Appendix D: Use case: introducing support for the OMF messaging system in the PlanetLab Europe testbed. All the considerations mentioned in this use case should be studied and addressed by the different testbed owners. This experience could prove to be a convenient starting point for other facilities that will have to introduce support for the OMF based Federated resource control protocol.

An initial **message syntax** for the federated resource control protocol can be found in Appendix E: Further NEPI specifications and examples. Here also requirements for installing NEPI and OMF6 EC can be found.

Experiment Control Engine API

Example of experiments using the OMF 6 EC can be found in [96] and [97].

The methods to be used in NEPI scripts to design and execute an experiment in case that the OMF6 FRCP protocol is applied are listed below:

create
Syntax: def create(self, factory_id, guid = None) **Functionality:**

Creates a new Box object.
See class TestbedDescription in src/nepi/core/design.py

Parameters:
factory_id: Box type identifier
guid: optional global unique identifier to set to the Box

Returns:
The box object of certain type, e.g.: Node

set_attribute_value

Syntax:
set_attribute_value(self, name, value)

Functionality:
Sets an attribute value in a Box object
See class Attribute in src/nepi/core/attributes.py

Parameters:
name: attribute name
value: attribute value

get_attribute_value

Syntax:
get_attribute_value(self, name)

Functionality:
Gets an attribute value in a Box object
See class Attribute in src/nepi/core/attributes.py

Parameters:
name: attribute name
value: value

Returns:
Attribute value

add_address

Syntax:
def add_address(self)

Functionality:

Add a network address to a Box that allows addresses
see class in UserAddressableMixin /src/nepi/core/factory.py

add_route

Syntax:
def add_route(self)

Functionality:
Add a routing entry to a Box that allows routes
see class UserRoutableMixin in /src/nepi/core/factory.py

connect

Syntax:
def connect(self, connector)

Functionality:
Establishes a connection between two connectors of different Boxes
see file src/nepi/core/design.py

Parameters:
connector: Connector object

enable_trace

Syntax:
def enable_trace(self, trace_id)

Functionality:
Activate a trace on a Box.
see class Box in src/nepi/core/design.py

start

Syntax:
def start(self)

Functionality:
Starts experiment execution. Launches deployment.
see class TestbedController in src/nepi/core/execute.py

stop
Syntax: def stop(self) **Functionality:** Stops experiment execution without releasing resources see class TestbedController in src/nepi/core/execute.py

shutdown
Syntax: def shutdown(self) **Functionality:** Shutdowns experiment controller releasing all resources see class TestbedController in src/nepi/core/execute.py

set
Syntax: def set(self, guid, name, value, time = TIME_NOW) **Functionality:** Sets resource attribute value during runtime see class TestbedController in src/nepi/core/execute.py **Parameters:** guid: resource id name: attribute name value: attribute value

get
Syntax: def get(self, guid, name, time = TIME_NOW) **Functionality:** Gets resource attribute value during runtime see class TestbedController in src/nepi/core/execute.py **Parameters:** guid: Global Unique Identifier of the resource name: attribute name

Returns: Resource attribute

status
Syntax: def status(self, guid) **Functionality:** Returns status of resource during runtime see class TestbedController in src/nepi/core/execute.py **Parameters:** guid: Global unique identifier of resource **Returns:** Status of resource

is_finished
Syntax: def is_finished(self, guid) **Functionality:** Determines wheather a resource has finished its task. See class TestbedController in src/nepi/core/execute.py **Parameters:** guid: Global unique identifier of resource **Returns:** Boolean value indicating weather the resource has finished its task or not

3.7.6 Requirements for testbeds to adopt the FRCP

Testbeds aiming to support this functionality should comply with the following requirements:

- Install the OMF 6 stable release as their testbed management software, or
- Implement as explained for PlanetLab their own version of the FRCP, to be able to control resources using the messages described above (create, inform, release, etc) and support the pub/sub messaging system
- Implement the corresponding resource controllers in case OMF 6 does not provide them, to be able to control their specific resources through an experiment controller

3.8 Support of existing experimenter front-ends and tools

Choosing SFA as the common control plane protocol allows to support the usage of existing SFA-compliant experimenter front-ends and tools. The next sections briefly list the existing tools that experimenters will also be able to use at the end of cycle 1. Note that the previous sections specified the different functional components of the Fed4FIRE architecture for the testbed, testbed management and broker levels. Together these levels can be considered to form the core Fed4FIRE architecture that this document has to specify. This section however is related to the experimenter level tools, which can be considered as optional components that will make use of the core architecture. Hence the need for detailed tools comparisons and specifications is less stringent in this case. This explains why the remainder of this section is structured differently than the previous sections of this chapter.

3.8.1 Teagle Framework Components: VCTTool and FCI

From the Teagle framework, as described in [105], two components are going to be supported. One the one hand the VCTTool that is used as a graphical user interface (GUI) for experimenters to specify the requested resources including their configuration parameters and interdependencies within a virtual customer testbed (VCT), see Figure 22. On the other hand the federation computing interface (FCI), which is further described in[106], to control the requested resources of a given VCT by generating C or Java classes, see Figure 23.

Currently, the VCTTool supports SFA by communicating with the Teagle core components that translate the requests to and responses from SFA-enabled domains by using an adapter. It is envisioned to integrate SFA support directly in the VCTTool in order to communicate with the according domains directly and to integrate the GUI as a plugin within MySlice. Although FCI currently supports SFA authentication and slice/resource browsing, further support is envisaged to manage slices through the FCI API (start/stop) and aggregate resources within a slice. Furthermore, FCI is planned to have a closer integration with the rest of the federated services by exploiting MySlice services.

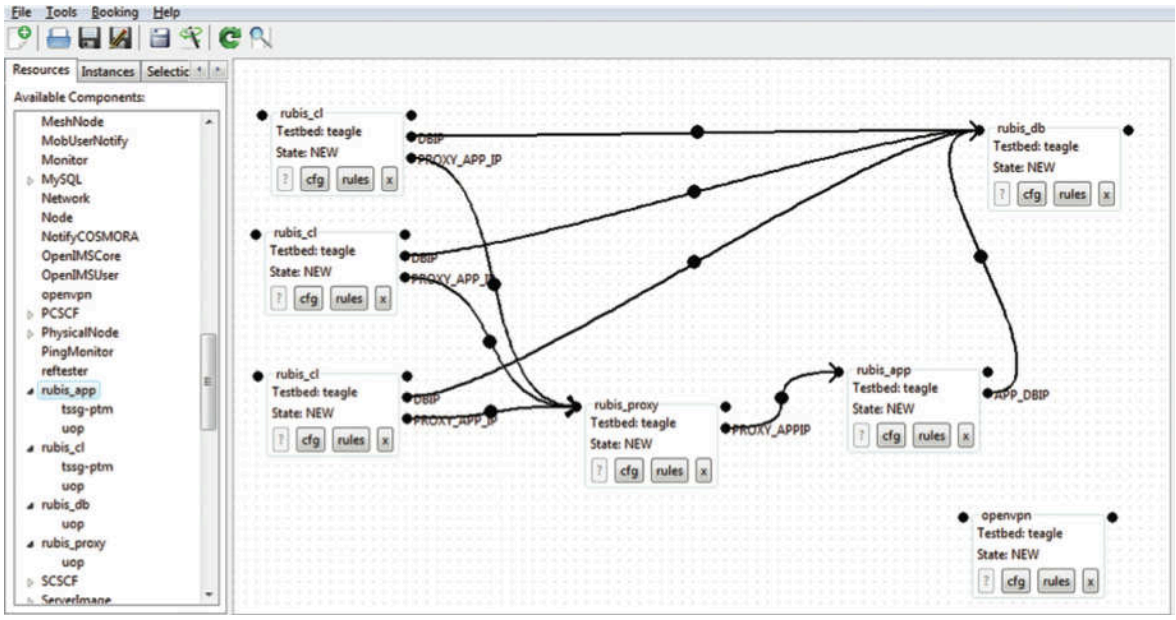

Figure 22: VCTTool with example VCT

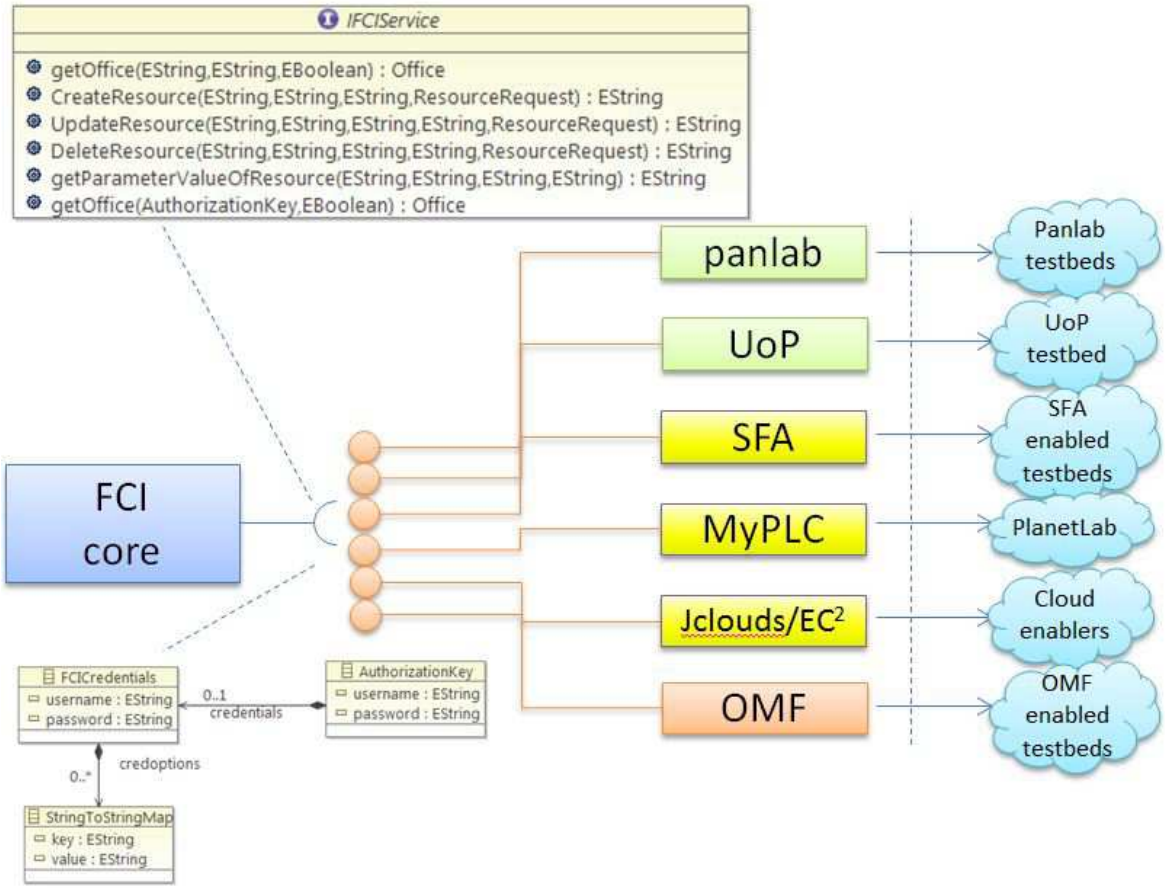

Figure 23: FCI Architecture Overview

3.8.2 Flack

Flack is a piece of software developed in the GENI project (USA) [12]. It is a visual client for ProtoGENI and federated GENI aggregate managers. Flack covers the main functionalities of authentication, discovery (example screenshots: Figure 24 and Figure 25) and resource provisioning over SFA compliant testbeds. It also allows the experimenter to create, open and update slices. It also provides the functionality to retrieve and display the utilized Rspec during different steps of the experiment lifecycle (Figure 26), which can be valuable throughout the development and integration process of Fed4FIRE.

In theory, Flack should be able to handle Fed4FIRE testbeds out of the box, since both rely on SFA. However, this will be verified at the moment that the testbeds have deployed their SFA interface during the course of development cycle 1. It is not inconceivable that Flack should be slightly updated to cope with possible implementation or versioning issues that emerge when Flack is tested against the different Fed4FIRE testbeds. If such needed small updates of Flack would be identified, then they will be implemented during cycle 1. This will allow the usage of the Flack tool together with the Fed4FIRE testbeds by the end of cycle 1.

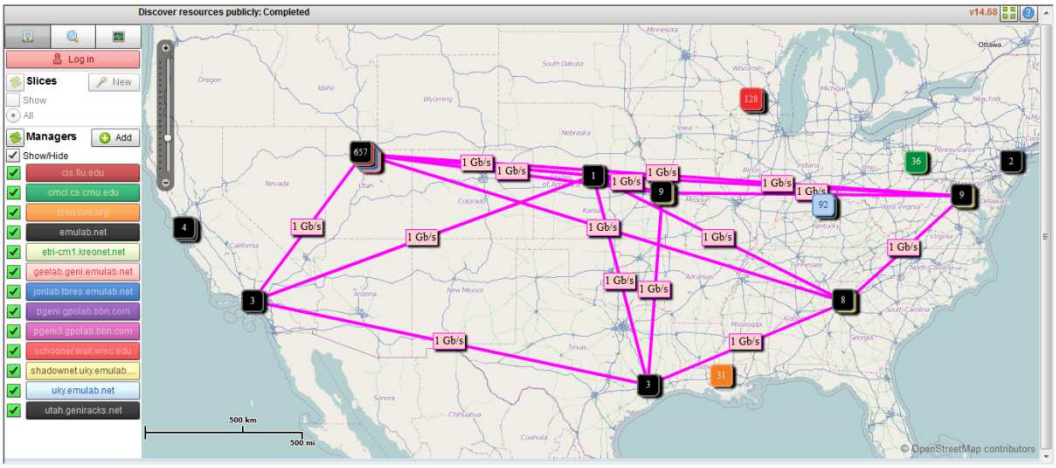

Figure 24: Flack - High level resource discovery using a map view

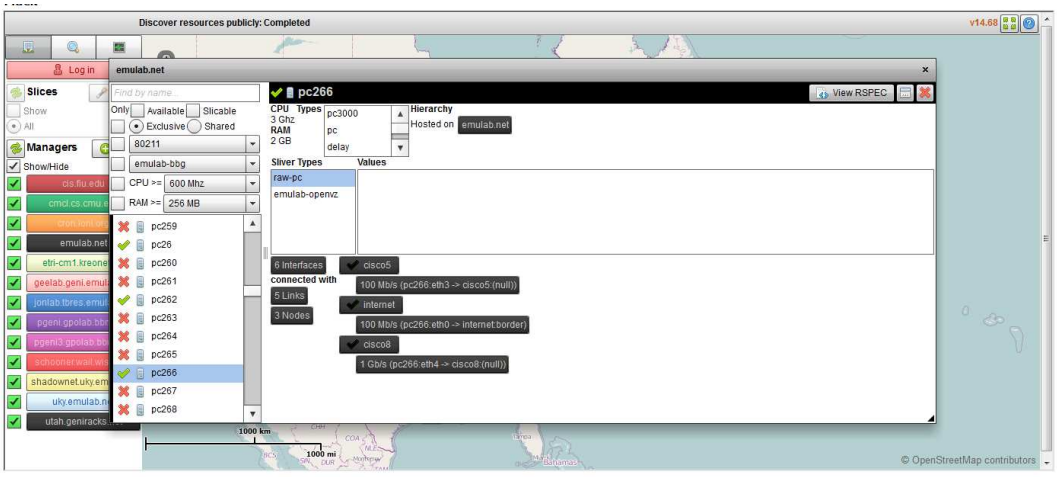

Figure 25: Flack - Detailed resource discovery per site

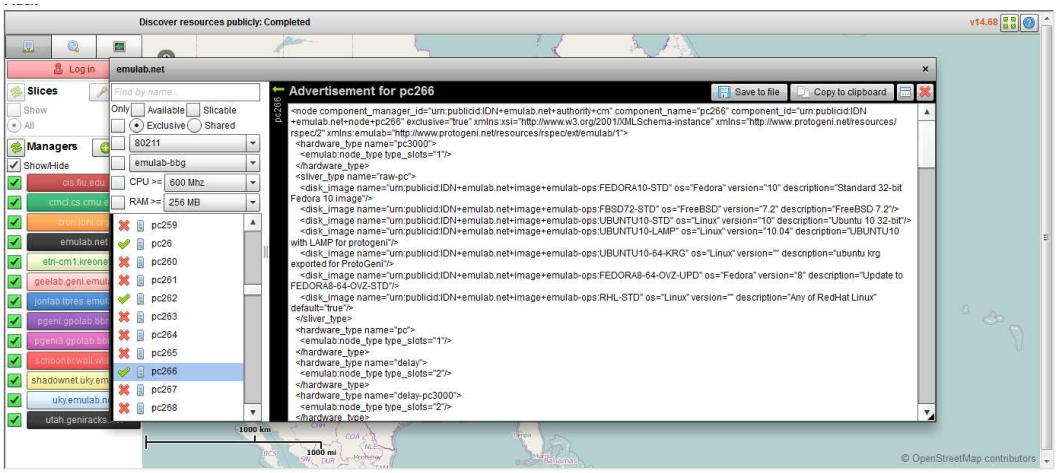

Figure 26: Flack - Immediate access to the applied Rspecs

3.8.3 Omni

Omni is a GENI command line tool for reserving resources at GENI Aggregate Managers (AMs) via the GENI AM API [107]. The Omni client also communicates with Clearinghouses (also known as Control Frameworks or CFs) to create slices, and enumerate available GENI AMs. A Clearinghouse is a framework of resources that provides users with GENI accounts (credentials). Users can use these credentials to reserve resources in GENI AMs. Any AM API compliant aggregate should work with Omni. These include SFA, ProtoGENI, OpenFlow and GCF.

```
$ omni.py createsliver aliceslice myRSpec.xml
INFO:omni:Loading config file omni_config
INFO:omni:Using control framework pgeni
INFO:omni:Slice urn:publicid:IDN+pgeni.gpolab.
          expires within 1 day on 2011-07-07
INFO:omni:Creating sliver(s) from rspec file
INFO:omni:Writing result of createsliver for
INFO:omni:Writing to 'aliceslice-manifest-rspe
INFO:omni: ------------------------------------
INFO:omni: Completed createsliver:

  Options as run:
                aggregate: https://www.emulab.
                framework: pgeni
                native: True

  Args: createsliver aliceslice myRSpec.xml

  Result Summary: Slice urn:publicid:IDN+pgeni
Reserved resources on https://www.emulab.net/p
  Saved createsliver results to aliceslice-man
INFO:omni: ====================================
```

Figure 27: Screenshot of the Omni command line tool

In theory (and similar to the Flack case), Omni should be able to handle Fed4FIRE testbeds out of the box, since both support SFA. However, this will be verified at the moment that the testbeds have deployed their SFA interface during the course of development cycle 1. It is not inconceivable that Omni should be slightly updated to cope with possible implementation or

versioning issues that emerge when it is tested against the different Fed4FIRE testbeds. If such needed small updates would be identified, then they will be implemented during cycle 1. This will allow the usage of the Omni tool together with the Fed4FIRE testbeds by the end of cycle 1.

3.8.4 SFI

SFI is another command line client for SFA interfaces [29] [109]. It is implemented in python as part of the (freely available) PlanetLab implementation. It provides the functionality to create, update and display a slice. SFI also supports resource discovery, reservation and provisioning. It can also be used to release resources, and to start and stop a slice. To illustrate how SFI is to be used, the help page of the tool is depicted in Figure 28.

In theory (and similar to the Flack and Omni cases), SFI should be able to handle Fed4FIRE testbeds out of the box, since both support SFA. However, this will be verified at the moment that the testbeds have deployed their SFA interface during the course of development cycle 1. It is not inconceivable that SFI should be slightly updated to cope with possible implementation or versioning issues that emerge when it is tested against the different Fed4FIRE testbeds. If such needed small updates would be identified, then they will be implemented during cycle 1. This will allow the usage of the SFI tool together with the Fed4FIRE testbeds by the end of cycle 1.

```
shell> sfi.py -h
Usage: sfi [options] command [command_options] [command_args]

Commands: list,show,remove,add,update,nodes,slices,resources,create,delete,sta
rt,stop,reset

Options:
  -h, --help            show this help message and exit
  -r URL, --registry=URL
                        root registry
  -s URL, --slicemgr=URL
                        slice manager
  -d PATH, --dir=PATH   config & working directory - default is
                        /Users/soltesz/.sfi/
  -u HRN, --user=HRN    user name
  -a HRN, --auth=HRN    authority name
  -v, --verbose         verbose mode
  -p PROTOCOL, --protocol=PROTOCOL
                        RPC protocol (xmlrpc or soap)
```

Figure 28: SFI - help page

4 Summary

This section provides a summary by first mapping the described solution against the architectural blue-print of D2.1 (First Federation Architecture) [1], first in the form of a table for functional mapping and second in the form of mapping components to the graphical representation of their architectural blue-print of D2.1. Finally the deviations from the original plan / architecture are described. The document finishes by providing an outlook on foreseen improvements.

4.1 Mapping of architecture to the implementation plan

In this section it is summarized how this deliverable transformed the architecture from D2.1 to an actual specification for implementation. This is done both in the form of a summary table, and by repeating the figures of the architecture, but with the adopted tools and clean slate implementations annotated in there.

Name functional element	Implementation strategy
Portal	Evolution of MySlice
Testbed directory	Extension of SFA API and MySlice Database, MySlice plugin
Tool directory	Extension of SFA API and MySlice Database, MySlice plugin, Wiki
Future reservation broker	Evolution of NITOS scheduler
Exposing testbeds through SFA	Initially evolution of SFAwrap, in the mid-term potential use of AMsoil
Experiment control	Cycle 1: FRCP and EC deployment on testbeds through OMF6 install. Cycle 2: add NEPI which interacts with the deployed FRCP layer.
Support of existing experimenter front-ends and tools	VCTTool and FCI, Flack, Omni, SFI

Table 22: Summary table mapping Fed4FIRE's architecture to selected solutions for cycle 1

The following Figure 29 maps the selected SFAWrap solution to Fed4FIRE's architecture requirement for testbeds to expose a common interface for resource discovery, resource reservation and resource provisioning.

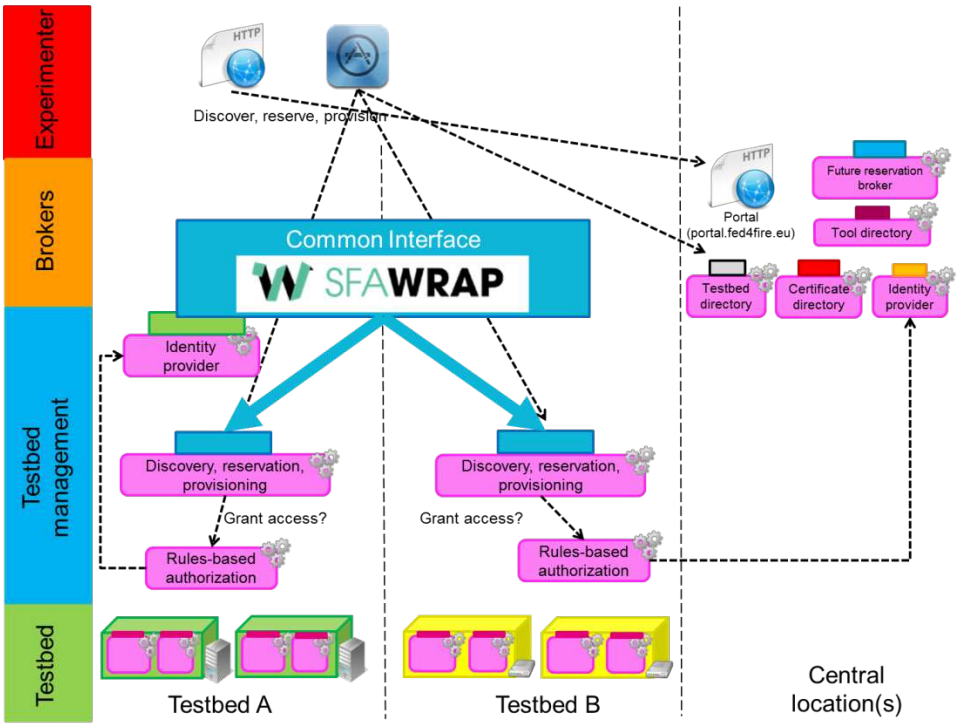

Figure 29: Fed4FIRE's cycle 1 approach for realizing a common testbed interface for resource discovery, reservation and provisioning

The following Figure 30 illustrates the selected solutions (MySlice as a basis for implementing Fed4FIRE's portal, NITOS scheduler for resource reservation, testbed and tool directory being an extensions of MySlice and the SFA API) for Fed4FIRE's architectural blue-print regarding resource discovery, resource reservation and registration.

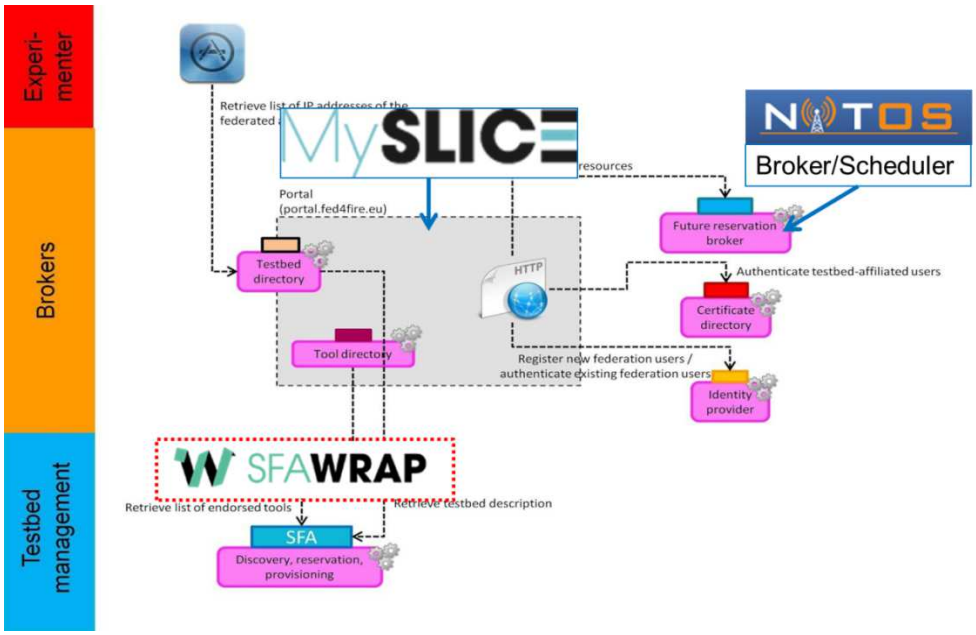

Figure 30: Fed4FIRE's cycle 1 approach for resource discovery, registration and reservation

Finally Figure 31 illustrates how the selected experiment control tools NEPI and the OMF Experiment Controller are mapped to Fed4FIRE's architectural "Experiment Control Server"

component. As a realization for the required common experiment control interface Figure 31 also shows the selected FRCP interface.

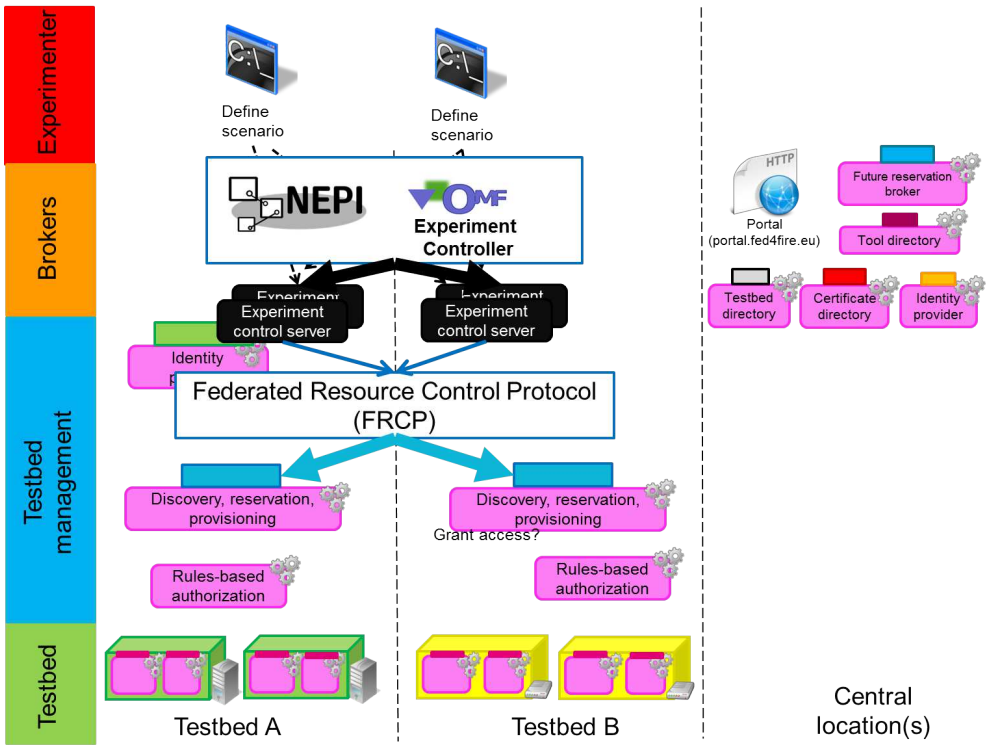

Figure 31: Fed4FIRE's cycle 1 approach for experiment control

4.2 Deviation of supported requirements compared to the architecture blue-print

At the point of writing there have been no decisions for implementation taken that would result in the fact that some requirements that D2.1 indicated are actually not supported anymore, or vice versa.

4.3 Foreseen improvements

This process of writing down the WP5 specifications has revealed that there is a bias in the current Fed4FIRE architecture towards infrastructures. Current experimenters of facilities are used to their own tools and this is something Fed4FIRE is respecting by not imposing any specific tool for experimentation. These users know the technologies these testbeds provide and are used to dealing with resources within these facilities. However, Fed4FIRE is bringing many heterogeneous testbeds together including very different technologies and it is unlikely that an average experimenter is familiar to all these at the same time.

That is why later iterations of this architecture should provide further utilities and services by introducing a new service layer for the experimenters. Based on a service-oriented architecture

(SOA), testbeds –and especially those considered in WP4– will be able to expose a range of services over the architecture, decreasing the complexity for the experimenter to directly deal with resources.

This vision can enormously 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.

For this purpose, testbeds must expose their available services as standards (such as WebServices, REST…), so that they can be discovered, negotiated as far as quality is concerned, invoked, monitored and accounted. Moreover, the possibility should exist for experimenters to select aggregated services, using services from different testbeds.

With this approach, aggregated services -composed of individually exposed services- would need to be orchestrated when the experiment is provisioned. For this, Fed4FIRE can introduce over this service layer new orchestration tools and engines, based on standards such as BPMN [99] or BPEL [100] (Activiti [102], jBPM [102], Apache ODE [103], Open ESB [104]…)

The tool should be an open, flexible and multiplatform solution for experimenters to design high-level composed processes/experiments. The services should be located in a repository where providers (different testbeds) can register and describe them semantically, using the ontology based in the first iteration of Fed4FIRE. The tool would suggest the most matching services as far as description is concerned for the experimenter to make a choice.

These new layers (service and orchestration) are not included in the first iteration and need to be placed in the architecture taking into account that some experimenters need not use services but resources, which means that we must find a way for both methods to be present at Fed4FIRE.

The following picture represents a preliminary idea of this concept.

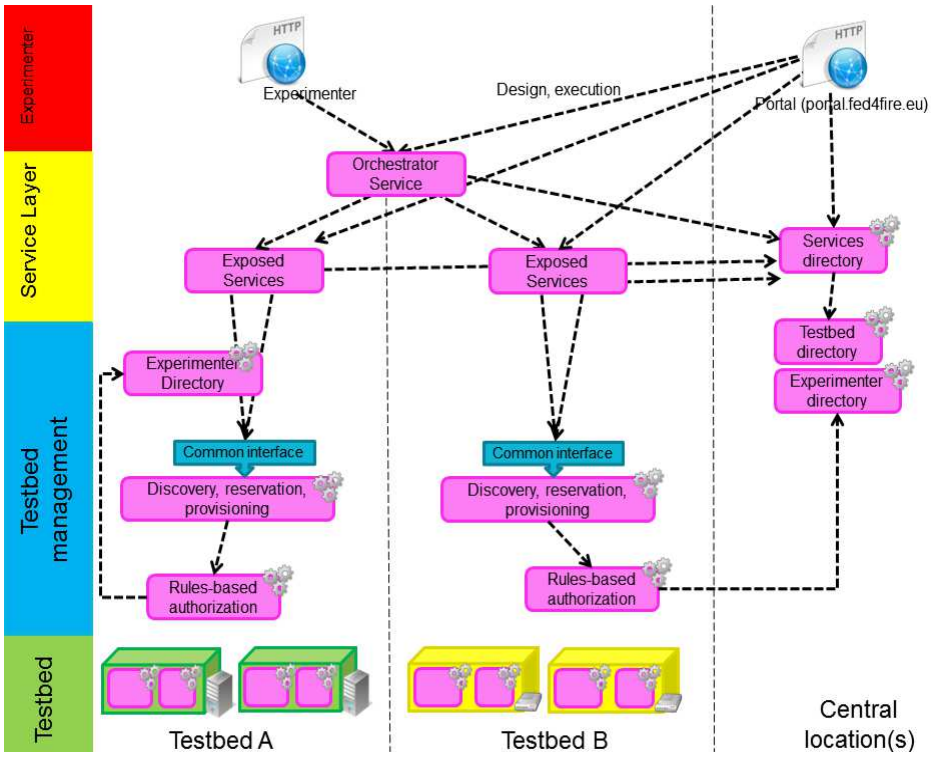

Figure 32: Possible future iteration of Fed4FIRE architecture

5 References

[1] Fed4FIRE D2.1 “First Federation Architecture”

[2] Fed4FIRE D3.1 “Infrastructures Community Federation Requirements”

[3] Fed4FIRE D4.1 “First Input from Community to Architecture”

[4] Fed4FIRE D8.1 “Fist Level Support”

[5] Larry Peterson, Robert Ricci, Aaron Falk, and Je_ Chase. Slice-Based Federation Architecture.Technical report, Geni, 2010. http://groups.geni.net/geni/wiki/SliceFedArch

[6] FITeagle Virtual Customer Testbed Tool. http://fiteagle.org

[7] Protogenie Emulab’s Flack tool. http://www.protogeni.net/ProtoGeni/wiki/FlackTutorial

[8] Network Experimentation Programming Interface, NEPI. http://nepihome.org

[9] LabWiki. “An Executable Paper Platform for Experiment-based Research”. http://www.nicta.com.au/pub?doc=4613

[10] BonFIRE Project. http://bonfire-project.eu

[11] BonFIRE Project’s Portal. http://doc.bonfire-project.eu/R3.1/client-tools/portal.html

[12] Geni’s FLACK tool. http://www.protogeni.net/wiki/Flack

[13] LabWiki. http://labwiki.mytestbed.net

[14] MySlice. http://myslice.info

[15] MySlice Overview. http://trac.myslice.info/wiki/MySliceOverview

[16] Manifold coordination language fo orchestration. http://projects.cwi.nl/manifold/

[17] Open Platform Architectures. Wikipedia. http://en.wikipedia.org/wiki/Open_platform

[18] TopHat. http://www.top-hat.info/

[19] MySlice Plugin Developer Guide. http://trac.myslice.info/wiki/PluginDeveloperGuide

[20] Joomla Content Management System. http://www.joomla.org/

[21] Django Content Management System. https://www.django-cms.org/

[22] MySlice Developer Website. http://trac.myslice.info

[23] MySlice Metadata Documentation. https://demo.myslice.info/components/com_tophat/css/images/myslice-metadata2.png

[24] MySlice Credential Delegation. http://trac.myslice.info/wiki/DelegatingCredentials

[25] SFA API Calls. http://groups.geni.net/geni/wiki/GAPI_AM_API_V3

[26] SFA API Return Struct. http://groups.geni.net/geni/wiki/GAPI_AM_API_V3/CommonConcepts#ReturnStruct

[27] SFA Aggreaget Manager API Version 3 Details. http://groups.geni.net/geni/wiki/GAPI_AM_API_V3_DETAILS

[28] Wiki Concept. http://en.wikipedia.org/wiki/Wiki

[29] SFA command line utility SFI. http://wiki.maxgigapop.net/twiki/bin/view/GENI/AccessingSlices

[30] GENI OMNI Experimenter Tool. http://trac.gpolab.bbn.com/gcf/wiki/OmniOverview

[31] JWiki – Joomla Example. http://extensions.joomla.org/extensions/social-web/social-edition/wiki-integration/12982
[32] Wikipedia Editiong Policy. http://en.wikipedia.org/wiki/Wikipedia:Editing_policy
[33] PlanetLab. http://www.planet-lab.org/
[34] NITOS testbed. http://nitlab.inf.uth.gr/NITlab/index.php/testbed
[35] VirtualWall. http://www.iminds.be/en/develop-test/ilab-t/virtual-wall
[36] iMinds w-ilab.t. http://www.iminds.be/en/develop-test/ilab-t/wireless-lab
[37] Future Seamless Communication Testbed - FuSeCo Playground. www.fuseco-playground.org
[38] NETMODE. http://www.netmode.ntua.gr
[39] Norbit Testbed at NICTA. https://mytestbed.net/projects/1/wiki/OMFatNICTA
[40] Grid 5000. https://www.grid5000.fr
[41] Smart Santander. www.smartsantander.eu
[42] Korean Future Internet Infrastrucutre Koren. http://www.koren.kr/koren/eng/index.html
[43] NITOS Scheduler. http://nitlab.inf.uth.gr/NITlab/index.php/scheduler
[44] OAR Resource and Job Management System. http://wiki-oar.imag.fr
[45] XML Remote Procedure Call. http://en.wikipedia.org/wiki/XML-RPC
[46] SFAWrap codebase. http://git.onelab.eu/?p=sfa.git;a=summary
[47] SFAWrap: developer tutorial. https://svn.planet-lab.org/wiki/SFADeveloperTutorial
[48] Sqlalchemy. the python sql toolkit and object relational mapper. http://www.sqlalchemy.org
[49] PlanetLab Europe. http://www.planet-lab.eu
[50] MyPLC User Guide. www.planet-lab.org/doc/myplc
[51] NITOS. http://nitlab.inf.uth.gr/NITlab/index.php/testbed
[52] SensLab. http://www.senslab.info/
[53] Federica. http://www.fp7-federica.eu/
[54] I2CAT Experimenta / OFELIA island. https://exp.i2cat.fp7-ofelia.eu
[55] FITeagle. http://fiteagle.org
[56] OSIMS. http://nam.ece.upatras.gr/ppe/?q=node/2
[57] SFA: Geni aggregate manager api version 2. http://groups.geni.net/geni/wiki/GAPI_AM_API_V2
[58] SFA: Geni aggregate manager api version 3. http://groups.geni.net/geni/wiki/GAPI_AM_API_V3
[59] OFELIA project. http://www.fp7-ofelia.eu
[60] SFAwrap. http://www.sfawrap.info
[61] GENI. http://www.geni.net
[62] AMsoil. https://github.com/fp7-ofelia/AMsoil
[63] OMF. https://mytestbed.net/projects/1/wiki/OMFatNICTA
[64] OMF Measurement Library OML. http://oml.mytestbed.net/projects/oml/wiki
[65] Zabbix Open Source Monitoring Solution. http://www.zabbix.com
[66] CollectD System Performance Statistics Collection Daemon. http://collectd.org/
[67] EmuLab. http://www.emulab.net/
[68] Teagle. http://trac.panlab.net/trac/wiki/

[69] Teagle Orchestration Engine. http://trac.panlab.net/trac/wiki/OrchestrationEngine
[70] BonFIRE. http://www.bonfire-project.eu/
[71] Omni. http://trac.gpolab.bbn.com/gcf/wiki/Omni
[72] OFELIA project: http://www.fp7-ofelia.eu/
[73] SFAwrap: http://sfawrap.info/
[74] AMSoil. https://github.com/fp7-ofelia/AMsoil
[75] SFA Test Suite. https://github.com/planetlab/sfa/tree/master/tests
[76] OMF Protocol Interactions. http://mytestbed.net/projects/omf/wiki/ArchitecturalFoundation2ProtocolInteractions
[77] NEPI Installation Guide. http://nepihome.org/wiki/InstallationGuide
[78] ProtoGenie. http://www.protogenie.com/
[79] Extensible Messaging and Presence Protocol (XMPP). http://en.wikipedia.org/wiki/XMPP
[80] RSpec. http://www.protogeni.net/ProtoGeni/wiki/RSpec
[81] Representational State Transfer REST. http://en.wikipedia.org/wiki/Representational_state_transfer
[82] Protogeni Credentials. http://www.protogeni.net/wiki/Credentials?searchterm=privilege
[83] A. Leivadeas, C. Papagianni, E. Paraskevas, G. Androulidakis, S. Papavassiliou, “An Architecture for Virtual Network Embedding in Wireless Systems”, IEEE First Symposium on Network Cloud Computing and Applications (IEEE NCCA 2011), Toulouse, France, November 2011.
[84] SFA Application Programming Interface. http://groups.geni.net/geni/wiki/GAPI_AM_API_V2_DETAILS
[85] OMF 6. https://omf.mytestbed.net/projects/omf/wiki/ArchitecturalFoundation2ProtocolInteractions
[86] OMF Architectural Foundation. https://mytestbed.net/projects/omf/wiki/Architectural_Foundation
[87] FITTeagle T1 interface specification. http://trac.panlab.net/trac/wiki
[88] BonFIRE Architecture. http://doc.bonfire-project.eu/R3/reference/bonfire-architecture.html
[89] ORCA Control Framework. https://geni-orca.renci.org
[90] Open Cloud Computing Interface OCCI. http://occi-wg.org/
[91] OMF Experiment Controller. http://omf.mytestbed.net
[92] OpenFire XMPP Server. http://www.igniterealtime.org/projects/openfire/
[93] NS-3 Network Simulator. http://www.nsnam.org/overview/what-is-ns-3/
[94] NEPI’s Network Experiment Frontend. http://nepi.pl.sophia.inria.fr/wiki/nef
[95] PlanetLab API. https://www.planet-lab.org/doc/plcapitut
[96] OMF Application Proxy. http://mytestbed.net/doc/omf/file.APPLICATION_PROXY.html
[97] OMF Resource Proxy. http://mytestbed.net/doc/omf/file.RESOURCE_PROXY.html
[98] Jabber / XMPP instant messaging server eJabberd. http://www.ejabberd.im/
[99] Business Process Model and Notation. http://en.wikipedia.org/wiki/Business_Process_Model_and_Notation

[100] Business Process Execution Language. http://en.wikipedia.org/wiki/Business_Process_Execution_Language
[101] Activiti BPM Platform. http://www.activiti.org
[102] Flexible Busienss Process Management Suite jBPM. http://www.jboss.org/jbpm
[103] Apache Orchestration Director Engine ODE. http://ode.apache.org
[104] Java-based Enterprise Service Bus OpenESB. http://www.open-esb.net
[105] Wahle, S., Tranoris, C., Denazis, S., Gavras, A., Koutsopoulos, K., Magedanz, T., & Tompros, S. . (2011). Emerging Testing Trends and the Panlab Enabling Infrastructure. IEEE Communications Magazine, 49(March), 167–175.
[106] Tranoris C., Denazis S. . (2010). Federation Computing: A pragmatic approach for the Future Internet», 6th IEEEInternational Conference on Network and Service Management, October 25-29, Niagara Falls, Canada.
[107] Omni. http://trac.gpolab.bbn.com/gcf/wiki/Omni
[108] Wikipedia Editing Policy. http://en.wikipedia.org/wiki/Wikipedia:Editing_policy
[109] SFI Tutorial. http:// www.cs.princeton.edu/~jrex/gew/gew-sfi.ppt

Appendix A: Development status of MySlice API

For the MySlice API, we adopted an object-oriented model, which has been proven to be general enough to accommodate for the need of the different interconnected platforms so far. A user will typically handle a collection, i.e. a set of objects. As shown in Figure 33 an object consists of a class (a type like slice, resource, traceroute, etc.), one or more fields, among which a key will uniquely identify the object instance, and zero or more methods. A property will eventually consist of a sub-collection (e.g. a slice holds a set of associated resources, a traceroute holds a set of hops, etc.).

Action	method	filters	params	fields	ts	callback
CREATE	✓			✓		!
GET	✓	✓		✓	✓	!
UPDATE	✓	✓	✓	✓		!
DELETE	✓	✓				!
EXECUTE	✓	✓	✓	✓		!

Figure 33: Development Status of MySlice API

Field consists of a name, a type/class, a flag to determine whether it is an array, a read-only or read-write flag, and a description.

Filters (the selection operator, or WHERE) consists of a clause of predicates. A predicate is a (key, operator, value) triplet, and a clause is a logical expression made from AND and OR operations separating clauses. Filters will allow us to build a collection.

Fields (the projection operator, or SELECT) enumerates a set of fields that allow to limit the scope of the objects being manipulated.

Params denotes a set of (key,value) pairs that can be used in queries (e.g. for updating fields is possible).

Timestamp (ts) and callback elements are optional and respectively allow to specify a time range for a query (timestamp, interval, keywords), as well as a return channel (for long, asynchronous, periodic or large queries, or to schedule notification of some events).

Description of Actions using Python notation:

Get
Syntax: Get(object, timestamp, filters, fields, callback)

Get
Functionality: Returns the values of an object Objects as defined by SFA: users, resources, authorities, slices **Parameters:** object: string timestamp filters: table of tables [[‘field’, ‘operator’, ‘value’]] fields: table of strings callback: function Returns: List of associative arrays

Update
Syntax: Update(object, filters, params, fields, callback) **Functionality:** Updates an object Objects as defined by SFA: users, resources, authorities, slices **Parameters:** object: string filters: table of tables [[‘field’, ‘operator’, ‘value’]] params: table fields: table of strings callback: function **Returns:** List of associative arrays

Create
Syntax: Create(object, params, callback)

Create
Functionality: Creates an object Objects as defined by SFA: users, resources, authorities, slices **Parameters:** object: string params: table callback: function **Returns:** List of associative arrays

Delete
Syntax: Delete(object, filters, callback) **Functionality:** Deletes an object Objects as defined by SFA: users, resources, authorities, slices **Parameters:** object: string filters: table of tables [[‘field’, ‘operator’, ‘value’]] callback: function **Returns:** List of associative arrays

Execute
Syntax: Execute(object, filters, params, callback) Functionality: Executes methods of an object Objects as defined by SFA: users, resources, authorities, slices

Execute
Parameters: object: string filters: table of tables [[‘field’, ‘operator’, ‘value’]] params: table callback: function **Returns:** List of associative arrays

Data Formats

Data formats used in the API such as filters or fields have to match the metadata provided by testbeds as properties.

As gateways will receive and process queries, they need to inform the upper layers about the functionality they realize by providing corresponding metadata. In other words, such metadata will give the minimum information about the part of the semantic schema they support, so that it can be efficiently used to route information correctly between platforms.

Such metadata will consist of the list of objects that are supported together with their properties and methods, a set of properties that can uniquely identify the object (keys), information indicating whether the property can be updated or is read-only, as well as some basic operators that can be applied to the data (and related to the relational algebra): sorting, windowing, grouping, etc

More detailed information can be obtained from MySlice’s Developer Website [22]. An example for auto-generated documentation about metadata as discussed above can be obtained here [23].

Appendix B: Further Specifications for SFAWrap Registry API Methods

GetVersion
Syntax: struct GetVersion([optional: struct options]) **Functionality:** Get static version and configuration information about this Registry. Return includes: • The version of the Registry API supported • The root authority HRN of the this Registry • The URL's of the peers root authorities **Parameters:** options: Optional **Returns:** a struct where the value member is Version Information

Register
Syntax: string Register(string caller_credentials[], struct object) **Functionality:** Registers an object (Authority, User, Slice) with the registry **Parameters:** • caller_credentials: The caller credentials • object: struct containing the object fields **Returns:** GID string representation

Update
Syntax: int Update (string caller_credentials[], struct object)

Functionality:
Updates an object (Authority, User, Slice) in the registry

Parameters:
- caller_credentials: The caller credentials
- object: struct containing the object updated fields

Returns:
'1' if successful, faults otherwise

Remove

Syntax:
int Remove(string caller_credentials[], string object_type, string object_xrn)

Functionality:
Removes an object (Authority, User, Slice) from the registry

Parameters:
- caller_credentials: The caller credentials
- object_type: Type of the object to remove
- object_xrn: The XRN of the object to remove

Returns:
'1' if successful, faults otherwise

Resolve

Syntax:
struct Resolve(string object_xrn, string caller_credentials[])

Functionality:
Used to learn the detailed information bound to a Registry object (Authority, User, Slice)

Parameters:
- object_xrn: The XRN (hrn or urn) of the object
- caller_credentials: The caller credentials

Returns:
a list of record dictionaries or empty list

List
Syntax: struct List(string authority_xrn, string caller_credentials[]) **Functionality:** Lists the set of Registry objects (Authority, User, Slice) managed by the named Authority **Parameters:** • authority_xrn: The XRN (hrn or urn) of the authority to list • caller_credentials: The caller credentials **Returns:** a list of record's HRN attached to that authority

GetSelfCredential
Syntax: string GetSelfCredential(string caller_cert, string object_xrn, string object_type) **Functionality:** a degenerate version of GetCredential used by the client to get his initial credential when he doesn't have one. **Parameters:** • caller_cert: Certificate of the caller • object_xrn: The XRN (hrn or urn) of the object • object_type: Type of the object (Authority, Slice, User) **Returns:** the string representation of a credential object

GetCredential
Syntax: string GetCredential(string object_xrn, string caller_credentials[], string object_type) **Functionality:** Retrieves the credentials corresponding to the named object (Authority, User, Slice)

Parameters:

- object_xrn: The XRN (hrn or urn) of the object
- caller_credentials: The caller credentials
- object_type: Type of the object (Authority, Slice, User)

Returns:

the string representation of a credential object

CreateGid

Syntax:

string CreateGid(string cert_owner_xrn, string caller_cert, string caller_credentials[])

Functionality:

Creates a signed certificate for the object with the registry

Parameters:

- cert_owner_xrn: The XRN of the certificate owner
- caller_cert: Certificate of the caller
- caller_credentials: The caller credentials

Returns:

GID string representation

Appendix C: Further AMSoil specifications and examples

This section explains in detail the specifications of the AMSoil tool, described in section 3.6.3 as well as an example to further clarify its structure and functionalities.

In Figure 34 we describe the scenario for a DHCP example.

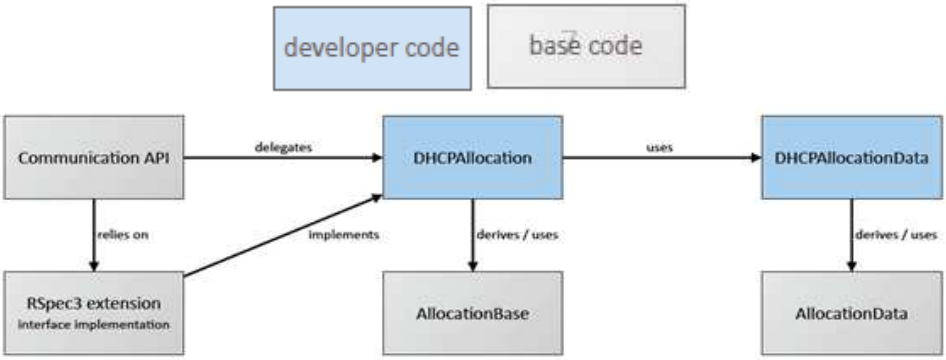

Figure 34: DHCP example

In that scheme we can see the base code, highlighted in grey, while the code implemented by the developer is highlighted in blue.

- The communication API provides an API Interface for clients.
- AllocationBase
    - Provides a base implementation for common tasks.
    - Implements / delegates persistence of basic data for allocation.
    - Does not assume any specification/communication protocols.
- Resource-specific implementations provide additional logic.
- Extensions provide standardized interfaces for optional, resource-specific logic.
    - At least one specification protocol is needed for instantiating an allocation.

The identification and authentication is implemented by the communication API. The AM developer is responsible for authorization and policy management. The communication API is already implemented in the base class, while the implementation of the policy manager is the responsibility of the developer.

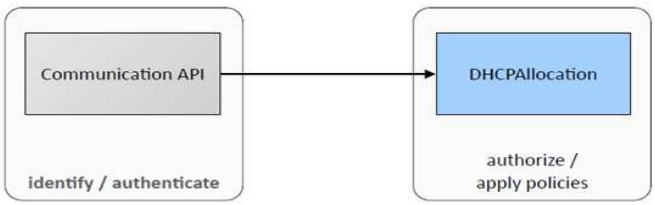

Figure 35: AMsoil – Relationship between Communication API and Policy Management

For additionally required implementations the AMsoil component delivers a series of base classes that the developer should extend in order to adapt it to its needs.

Specifications for AMsoil

Most core and custom functions of AMsoil are encapsulated in plugins. Each plugin may provide services, which encapsulate the actual functionality. A plugin consists of the following three things:

- A `MANIFEST.json` file, specifying the services and dependencies of the plugin
- A `plugin.py` file which contains a `setup()` method for initialization and registering services
- The actual implementation of the plugin

Example for a `MANIFEST.json`:

```
{
  "name" : "RPC Registry",
  "author" : "Tom Rothe",
  "author-email" : "tom.rothe@eict.de",
  "version" : 1,
  "implements" : ["rpcserver", "xmlrpc"],
  "loads-after" : ["config"],
  "requires" : []
}
```

In order to use a plugin, the name of the required service must be added in the manifest file. If the setup method (or an import in the plugin module) requires the service `loads-after` must be used. Or if the plugin does only need the service after initialization then `requires` is used. After that, a reference to the service must be got and methods which are published must be called:

```
import amsoil.core.pluginmanager as pm
xmlrpc = pm.getService('xmlrpc')
xmlrpc.registerXMLRPC('geni2', GENIv2RPC(), '/RPC2')
```

It is noted, that methods which can be used from a plugin are usually marked with the decorator `@serviceinterface`.
Consider the following example of a plugin implementation:

```
class FlaskXMLRPC(object):
      @serviceinterface
      def registerXMLRPC(self, unique_service_name, instance,
endpoint):
       ...
```

Which has been registered in the plugin under the name xmlrpc:

```
import amsoil.core.pluginmanager as pm
xmlrpc = FlaskXMLRPC(flaskserver)
pm.registerService('xmlrpc', xmlrpc)
```

While writing plugins for the AMsoil, the tool developer may bind (`registerService(...)`) basically everything to a service name. This may be an object, class, dictionary or even a module. So create a new folder in plugins, create the manifest (be sure to put all services to be implemented in the `implements` section), `plugin.py` and the implementation. Then all that is needed is to call is `registerService(...)` in `setup` method and annotate the things wanted to be used with `@serviceinterface`.

Appendix D: Use case: introducing support for the OMF messaging system in the PlanetLab Europe testbed

In this appendix previous efforts are described that resulted in the introduction of support for the OMF messaging system in the PlanetLab Europe testbed. In this example the current stable version 5.4 of OMF was applied, with the final goal of supporting the OMF messaging system being to allow the control of PlanetLab resources using the OMF EC.

PlanetLab testbeds consist in providing bare resources as virtual machines, and in leaving experiment control to third-party tools. The control and management software for PlanetLab is called MyPLC.

As explained before the OMF PubSub system enables the communication between the EC (user front-end) and RCs (Resource Controller). This results in four sets of requirements, in order to make a PlanetLab deployment accessible to an EC:

- the MyPLC software must be able to create and maintain the topics that correspond to the various entities in the system, and namely nodes, and slices;
- for this to be possible at all, there is a need for an XMPP-server to run alongside the MyPLC;
- there is a need for an instance of the RC to be present in the virtual machines (named slivers in PlanetLab) that might be controlled through the EC;
- finally, there is a need to assess, and if necessary provide, security mechanisms so that only entitled persons can take control of any given sliver.

Maintaining topics :

It is beneficial to keep the interactions between MyPLC and XMPP as clean as possible, notably because XMPP, being deeply asynchronous, often leads to convoluted programming techniques. For that reason it was decided to run a separate service, which comes with the MyPLC software - and that behaves as a proxy between the MyPLC infrastructure and its related XMPP server. This service offers an XML-RPC interface to the MyPLC side, and in turn issues the relevant XMPP commands to the PubSub server.

Running an XMPP server and MyPLC configuration :

There is no strong nor compelling reason for having as many XMPP servers as there are testbeds; as a matter of fact it is quite doable – and occasionally the system has been used this way during the early stages – to have a single XMPP server host all the topics that belong in several testbeds. However, this obviously is not a very scalable approach, and our general recommendation is to have each testbed operations role out their own XMPP server, that can then peer together much like SMTP servers would.

This does not have a significant impact on the MyPLC software itself, since it is quite possible to deploy any standard XMPP implementation - provided that it supports topics for group

communication. This support of topics now tends to be the norm but it was not the case a couple of years ago, so the use of a recent release is highly recommended.
PlanetLab experimented with 2 major implementations:

- ejabberd is an implementation written in erlang. For PlanetLab there was not any particular issue while using ejabberd, but contacts at NICTA, who run their own OMF-based testbed, have reported issues when scaling up with the number of messages;
- this is the reason why PlanetLab eventually opted for openfire, which is a java implementation that currently runs under xmpp.planet-lab.eu, the XMPP server that is used as a companion to PlanetLab Europe.

Deploying RCs:

The whole point in deploying XMPP is to have the EC send messages to the resources to be controlled, in the PlanetLab case virtual machines, or slivers.
For the whole communication scheme to be completed though, it is also needed to ensure that messages will correctly reach their destination. This destination is an instance of a RC on the target slivers.
As a general rule, the PlanetLab software tries very hard to not interfere with the running slivers, but this was considered a valuable exception where the infrastructure could fruitfully implement additional services, that is to say, take care of pre-installing and pre-configuring a working instance of the RC, so that experimenters do not have to worry about anything and can use the EC right away after having created their slivers using the MyPLC API. All this is done only when explicitly requested so at slice-creation time, so that only users who want the RC service will have it running on their slivers.
In terms of the installed software, ruby code as published by OMF is used, it is only re-packaged for convenience. Service configuration is performed by a dedicated plugin to the regular PlanetLab node manager, that is completely straightforward as it is essentially a matter of exposing the address of the XMPP SERVER, as configured in PLC OMF XMPP SERVER, to the RC configuration file.

Communication Security:

The way the OMF EC was usually deployed was to be run from inside the OMF "gateway". This gateway is a simple yet effective way to implement security, in the sense that users receiving an account would essentially get a login account on this gateway, which then allows them to access the real OMF resources, that otherwise have no general Internet connectivity. All this had lead to the fact that, in a purely OMF context, there was no need to implement any security mechanism in the XMPP exchanging mechanism, because essentially people in position to use the XMPP service at all had already been authenticated.
For PL characteristics this assumption does not hold anymore, and PL had to come up with the following scheme for making the system secure:

- The native XMPP authentication scheme was considered, but rejected for PL needs, because it is both weak and adds extra complexity;
- rather, it was decided to perform an end-to-end encryption, based on ssh keys because they are already present in the PlanetLab authentication scheme;

- changes had been made to the PlanetLab software, so that the authenticated keys for each sliver become visible from the sliver itself; they were not, and it turns out this was a very welcome feature for people who did not care about OMF as well. This way the RC can make sure that the incoming message truly has been signed with a private key that matches those of the people entitled to use the sliver;
- and, the other way around, since a resource may have to send messages back to the EC, each sliver that is “OMF-friendly” generates its own keypair, and exposes the corresponding public key through the PLCAPI method GetSliceSshKeys2 , so the EC can also check the authenticity of messages coming back from resources.

Appendix E: Further NEPI specifications and examples

This section first explains the message syntax for the federated resource control protocol (FRCP). For the experiment control engine, the methods to support the new design and architecture are listed in section 3.7.5.

The content of a message is described in an XML format with the following convention.

```
<MSG_NAME xmlns="http://schema.mytestbed.net/omf/X.Y/protocol"
msg_id= ID>
....
</MSG_NAME>
```

with:

- X.Y = the version of the protocol
- MSG_NAME = the name of message, either "create", "configure", "request", "inform", or "release"
- ID = a globally unique ID for this message

The element may then have child elements, which further describe various message properties specific to the message type. We define two different ways to declare the value of a message property. The simple version provides the value as a text element with an optional 'type' attribute if the value is not of type 'xsd:string'.

```
<MSG_NAME ... >
<PROP_NAME type="TYPE">VALUE</PROP_NAME>
...
</MSG_NAME>
```

The more descriptive way employs a list of child elements to describe the property in more details. Basic elements are 'type' and 'value', but can also include 'unit', 'precision', or 'min-value', 'max-value' if the property is used a constraint in a 'request' message.

```
<MSG_NAME ... >
<PROP_NAME>
<type>TYPE</type>
<value>VALUE</value>
...
</PROP_NAME>
...
</MSG_NAME>
```

The specific parameters/child elements are described below for each message type.
In addition to these specific parameter/child elements, a message may also carry an optional 'guard' element, which carries constraint information. When a such a guard element is present, an entity will only act on a received message if it satisfies the described constraints.

Create Message

A create message is sent to a resource to ask it to create another resource. The creator is referred to as the parent (P) and the newly created resource as the child (C). This message may contain optional pairs of (key, value), if so then the created resource C should have its property key to the given value.

Parameters

- resource_type = the type of child resource to create, valid existing type could be queried from the parent resource or discovered out-of-band
- (k, v) = the key for the property 'k' to set for the child resource and the associated value 'v' to use
- ...

Behaviour

1. Upon receiving a create message, the resource P must decide if it will create the requested child resource. This decision may requires P to check if it is technical capable of creating the type of resource requested, and/or it its governing policy allows it to do so.
2. If P decides to create C, then P must select a globally unique ID for C, map this globally unique ID to a new topic address (using the previously described mapping convention), and create that respective topic in the pubsub system
3. P must then allocate the new resource C, depending on C's type this could involve tasks such as starting up a new VM, starting up a new application, activating a device/interface. Our proposed architecture does not provide any guarantee on this allocation process, i.e. it could take an arbitrary duration, it could fail for x reasons, etc... However nothing prevents institutions providing resources to offer such kind of guarantees
4. P must publish an inform message to its own topic to provide feedbacks on the creation request. The inform message must have the following parameters:
5. context_id = the ID of the original create message related to this inform message
6. inform_type = the type of inform message, either "CREATION_OK" or "CREATION_FAILED"
7. resource_id = the globally unique ID of the newly created resource C
8. resource_topic = the topic address of the newly created resource C

Example for creating a virtual machine:

```
<create xmlns="http://schema.mytestbed.net/omf/6.0/protocol"
message_id="1ab3f0">
      <resource_type>virtual_machine</resource_type>
            <property key="vm_type">xen</property>
            <property key="os" type="string">ubuntu-
      11.10</property>
      <property>
      <key>memory</key>
      <type>fixnum</type>
      <value>2</value>
      <unit>GB</unit>
      </property>
</create>
```

```
<inform xmlns="http://omf.mytestbed.net/omf/6.0/protocol"
message_id="27ab23">
      <context_id>1ab3f0</context_id>
      <inform_type>CREATION_OK</inform_type>
      <resource_id>foo</resource_id>
      <resource_address type="xmpp">foo@bar</resource_address>
</inform>
```

Starting an application

```
<publish node='vm1'>
      <create xmlns=http://schema.mytestbed.net/omf/6.0/protocol
message_id="962ac5">
            <resource_type>application</resource_type>
            <property key="path">/bin/ls</property>
            <property key="option">-la</property>
      </create>
</publish>
```

Configure Message

A configure message is sent to a resource to ask it to set some of its properties to some given values.

Parameters:

- (k1, v1) = the key for the 1st property to set and the associated value to use
- ...
- (ki, vi) = the key for the i-th property to set and the associated value to use

Behaviour:

- Upon receiving a configure message, the resource should try to set its property named key to the given value for each pair of (key, value) included in the message.
- The resource must publish an inform message to its own topic (i.e. the topic derived from its unique resource ID), which provides feedbacks on the configure request
  - that inform message must have the following parameters:
    - context_id = the ID of the original configure message related to this inform message
    - inform_type = the type of inform message: "STATUS"
    - the list of properties to configure and their required values. Each item on that list may be either a simple structure as:
  - key => name = the name of the property, e.g. k1
  - value => value = the new value for this property
  - or an optional extended structure as:
  - key => name = the name of the property, e.g. k1
  - current => value = the current value of the property
  - target => value = the target value for the property, this is the value requested in the last configure message which included this property

- msg => string = an optional text message giving additional information on the property or its setting, e.g. success, error, ongonig, etc...
- progress => integer = an optional value from [0,100] indicating the progress of the property setting (with 100 = completed)

Note about a resource's state:

- As described below, the current state of the resource is held in a specific property named 'state'
- This 'state' name is reserved and must not be used to identify any other properties of a resource
- The state property could have the top-level values "ACTIVE" or "INACTIVE", or any resource-specific values corresponding to resource-specific sub states, such as "ACTIVE/RUNNING", "ACTIVE/PAUSED", etc...

Example:

```
<configure xmlns="http://schema.mytestbed.net/omf/6.0/protocol"
msg_id="37d76f">
<property key="bitrate" type="string">512</property>
<property key="protocol" type="fixnum">UDP</property>
<property key="pkt_size" type="string">1024</property>
</configure>
<configure xmlns="http://schema.mytestbed.net/omf/6.0/protocol"
msg_id="3a81e0">
<property key="state" type="string">ACTIVE/PAUSED</property>
</configure>
<inform xmlns="http://schema.mytestbed.net/omf/6.0/protocol"
msg_id="2e6c38">
<context_id>3a81e0</context_id>
<inform_type>STATUS</inform_type>
<property key="state" type="string">ACTIVE/PAUSED</property>
</inform>
```

Optional Extended Inform Format:

```
<inform xmlns="http://schema.mytestbed.net/omf/6.0/protocol"
msg_id="2e6c38">
<context_id>3a81e0</context_id>
<inform_type>STATUS</inform_type>
<property key="state" type="string">
<current>ACTIVE/RUNNING</current>
<target>ACTIVE/PAUSED</target>
<msg>switching state in progress</msg>
<progress>70</progress>
</property>
</inform>
```

Request Message:

- A request message is sent to a resource to ask it to publish some information about some of its properties.
- Parameters:
  - publish_to = an optional name of an additional topic address to which the reply message should be published to
  - k1 = the key for the 1st property for which information is requested
  - ...
  - ki = the key for the i-th property for which information is requested

Behaviour:

Upon receiving a request message, the resource should publish an inform message, which provides information on the properties included in the request message

This inform message should be published to the resource's own topic (i.e. the topic derived from its unique resource ID) by default.

If the publish_to parameter is set in the original request message, then the replying inform should also be published to the topic address given in that parameter

That inform message must have the same format as the inform message published in reply to a configure message.

- Example:

```
<request xmlns="http://schema.mytestbed.net/omf/6.0/protocol"
msg_id="ea4afb">
<property key="bitrate" />
<property key="protocol" />
<property key="pkt_size" />
</request>
<request xmlns="http://schema.mytestbed.net/omf/6.0/protocol"
msg_id="3432ea">
<publish_to>foo@bar</publish_to>
<property key="state" />
</request>
<inform xmlns="http://schema.mytestbed.net/omf/6.0/protocol"
msg_id="fd3e77">
<context_id>3432ea</context_id>
<inform_type>STATUS</message_type>
<property key="state" type="string">ACTIVE</property>
</inform>
```

Release Message

A release message is sent to a resource to ask it to release (i.e. terminate) one of its resource children.

- Parameters:

resource_id = the globally unique ID of the resource children to release

Behaviour:

- Upon receiving a release message with the ID of one of its child, a resource must contact that child and instruct it that it will be imminently released. This gives the opportunity for the child resource to perform any cleanup tasks if required.
- Once the child is ready to be released, the parent resource must initiate the child's termination and remove its corresponding pubsub topic.
- Once the child has been successfully released, the parents must publish an inform message to its own topic to confirm the child's release
- that inform message must have the following parameters:
- context_id = the ID of the original release message related to this inform message
- inform_type = the type of inform message: "RELEASED"
- resource_id = the globally unique ID of the child resource which was released
    - Important Note: Any children of the child resource currently being released must all be released first! Thus once a resource receives a release instruction from its parent, it must releases all of its own children resource before notifying its parent that it is itself ready to be released.

Example, the following release message is sent to the pubsub topic of the parent of the resource "foo":

```
<release xmlns="http://schema.mytestbed.net/omf/6.0/protocol"
msg_id="806a7e">
<resource_id>foo_resource</resource_id>
</release>
<inform xmlns="http://schema.mytestbed.net/omf/6.0/protocol"
msg_id="5d5a46">
<context_id>806a7e</context_id>
<inform_type>RELEASED</inform_type>
<resource_id>foo_resource</resource_id>
</inform>
```

Inform Message

An inform message is published by a resource wishing to provide some information about some of its properties, or about some previous tasks that it was instructed to do.

Parameters:

- context_id = optional, if this inform message is a reply to another message, then this must be set to the ID of that original message
- inform_type = the type of this inform message, either "CREATION_OK", "CREATION_FAILED", "STATUS", "RELEASED", "ERROR" or "WARN"
- additional parameters depending on the type of this message

see above create message for parameters related to "CREATION_OK" or "CREATION_FAILED" inform messages

see above configure message for parameters related to "STATUS" inform message

see above release message for parameters related to "RELEASED" inform message

"ERROR" and "WARN" are used for sending generic error or warning inform messages, for example to report errors generated from application execution. Can use "reason" parameter to provide some useful error/warning explanation.

Behaviour:

- A resource may publish an inform message spontaneously to report about some of its properties
- A resource must publish an inform message as a reply to previously received create, configure, request, or release messages
- A resource must publish its inform message on the topic address mapped from its globally unique ID. If this inform message is a reply to a previous message with a publish_to field set in it, then a copy of this inform message must also be publish to the topic address specified in that field.

Examples: see above examples for other message types

Optional Guard Element

All the messages described above may also include an optional 'guard' element. When present, this element contains itself a Hash of properties and values (implemented as sub-elements similar to the above configure message). When an entity receives a message with a guard element, it must act on the message (according to the above described behaviours for each message) ONLY if all of its properties mentioned in the guard element have the same values as the one mentioned in the guard element.

Thus, if a guard element contains a Hash with two properties and their corresponding values p1=v1 and p2=v2, the message recipient will only act on the message if it has a p1 property of value v1 and a p2 property of value v2. If not, it will discard the message.

Example of a Configure Message with a guard element:

```
<configure xmlns="http://schema.mytestbed.net/omf/6.0/protocol"
msg_id="37d76f">
<property key="bitrate" type="integer">512</property>
<property key="protocol" type="string">UDP</property>
<property key="pkt_size" type="integer">1024</property>
<guard>
<propery key='protocol' type='string'>UDP</property>
<propery key='state', type='string'>running</property>
</guard>
</configure>
```

Requirements for installing NEPI and OMF6 EC

To install OMF 6 in testbeds this is the dependencies the testbed should support:

- Ruby 1.9 to be installed on the hosts running the resource proxies and the experiment loader
- OMF 6 requieres Ruby Gems
- For the XMPP-based Publish-and-Subscribe the recommended server is OpenFire (>3.7.1)

In the case of NEPI experiment controller, the dependencies below must be met on the system prior to installing NEPI and its modules:

- NEPI requires:

  - python >= 2.6
  - ipaddr >= 2.1.5
- NETNS requires:
  - Linux kernel >= 2.6.36
  - bridge-utils
  - iproute
- NEF requires:
  - libqt4 >= 4.6.3
  - python-qt4 >= 4.7.3
- ns-3 requires:
  - python-dev

The installation of OMF 6 includes the OMF EC, therefore, fulfilling the ruby dependencies described above is enough for running the experiment controller.