Facility provider training

Wim Vandenberghe - Piet Demeester

May 7th 2013

Intended audience

- Facility providers interested in joining the federation in the future.
- Fed4FIRE will fund two open calls for such additional facilities.
  - Will be launched in May 2013 & 2014
  - Open to academia and industry
  - Selected proposals: will join the project until the end of the project
  - Max. funding 100 k EUR / proposal

FED4FIRE

iMinds CONNECT.INNOVATE.CREATE

Outline

- What is Fed4FIRE?
- Overview of the Fed4FIRE architecture
- Implementation of the Fed4FIRE architecture
- Implications of joining Fed4FIRE on the testbeds

What is Fed4FIRE?

- FP7 IP project: Federation for FIRE
- www.fed4fire.eu
- Project summary: Fed4FIRE will bring a common federation framework for Future Internet Research and Experiment facilities that will
  - be widely adopted by different communities (experimentation facilities, experimenters, academia, industry)
  - support powerful experiment lifecycle management (including tools for discovery and reservation, experiment control, measurements, etc.)
  - support key aspects of trustworthiness (federated identity management and access control, accountability, SLA management)

What's in it for you (the facility provider)?

- Reach a larger community of potential experimenters through
  - common resource discovery mechanisms
  - common dissemination activities
- Increase the technical possibilities for your experimenters
  - Access to other testbeds in the Fed4FIRE community
  - Support of multi-testbed experiments
- Lower your costs through
  - The adoption of jointly developed common tools
  - The participation in a common First Level Support service

FED4FIRE

iMinds CONNECT.INNOVATE.CREATE

Fed4FIRE – general info

- Federation for FIRE
- IP project
- 10/2012 - 9/2016

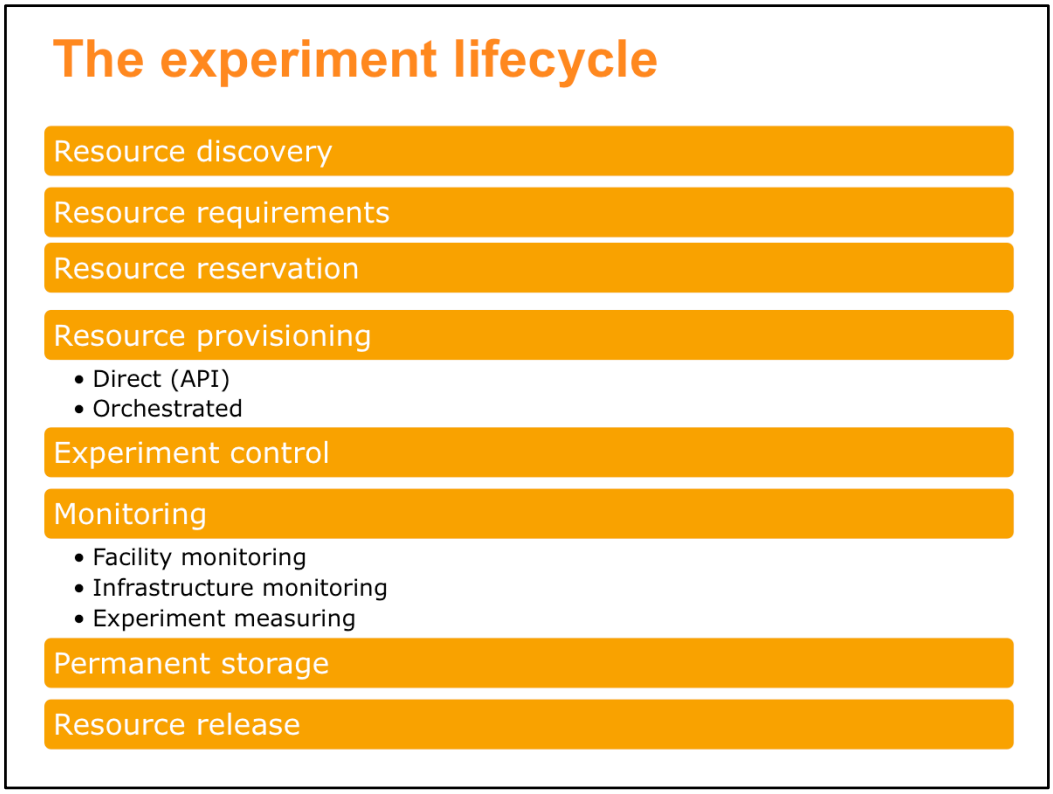
- project coordinated by iMinds
- Total budget: 7.75 MEUR

Project partners

Experimentation Facilities

Outline

- What is Fed4FIRE?
- Overview of the Fed4FIRE architecture
- Implementation of the Fed4FIRE architecture
- Implications of joining Fed4FIRE on the testbeds



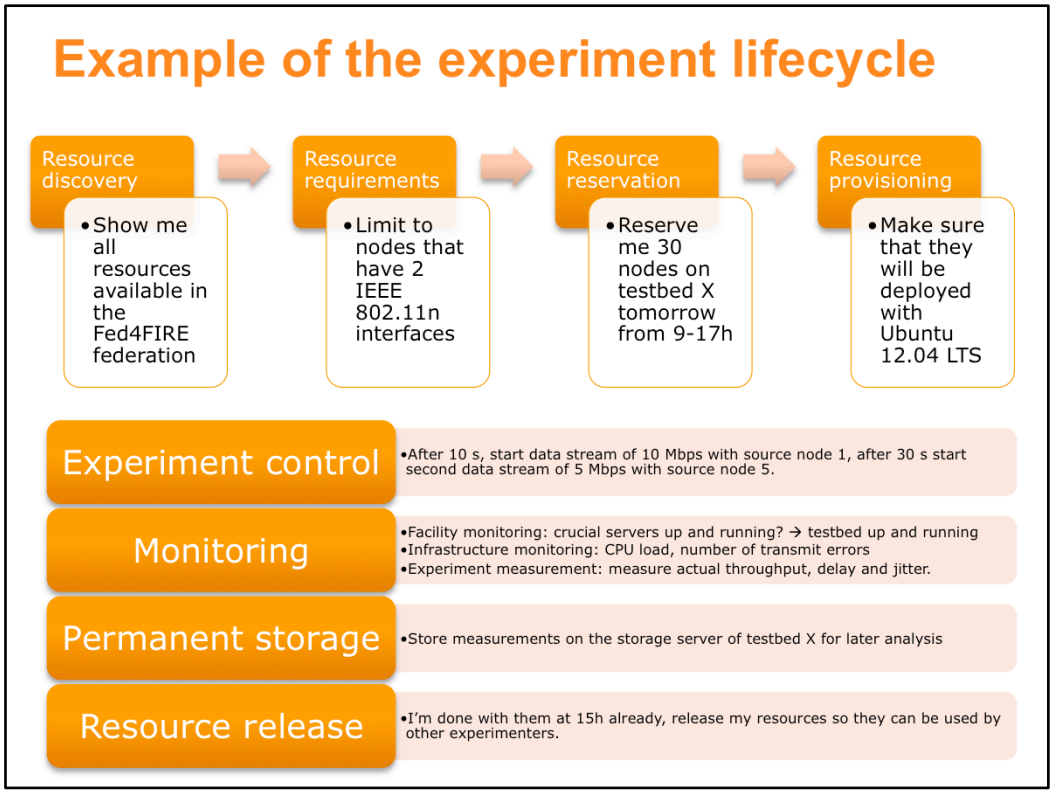
Resource discovery: Finding available resources across all facilities, and acquiring the necessary information to match required specifications.
Resource requirements: Specification of the resources required during the experiment, including compute, network, storage and software libraries.
Resource reservation: Allocation of a time slot in which exclusive access and control of particular resources is granted.
Resource provisioning

Direct (API): Instantiation of specific resources directly through the facility API, being the responsibility of the experimenter to select individual resources.

Orchestrated: Instantiation of resources through a functional component, which automatically chooses resources that best fit the experimenter's requirements.
Experiment control: Control of resource behavior during experiment execution, involving actions to query and modify resource state, and their correct sequencing.
Monitoring

Facility monitoring: Instrumentation of resources to supervise the behavior and performance of facilities allow system administrators or first level support operators to verify that facilities are performing correctly.

Infrastructure monitoring: Instrumentation of resources to collect data on the behavior and performance of services, technologies, and protocols to obtain measurements in the context of a concrete experiment.
Experiment measuring: Collection of experiment data generated by frameworks or services that the experimenter can deploy on its own.
Permanent storage: Storage of experiment related information beyond the experiment lifetime, such as experiment description, disk images and measurements.
Resource release: Release of experiment resources after deletion or expiration the experiment.



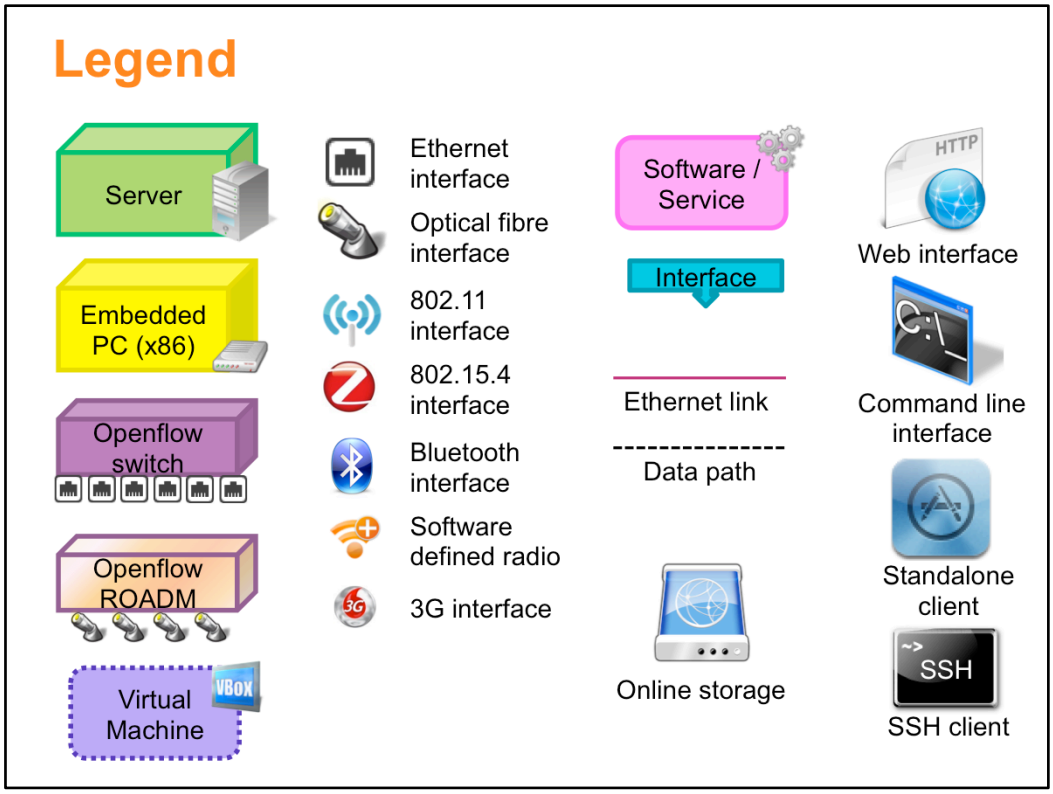
In this example an experimenter has developed a mechanism to automatically create a wifi mesh network (multi-hop network). The experimenter wants to test this at a larger scale, hoping to proof that the new solution can easily forward multiple streams at the same time without sacrificing any performance.

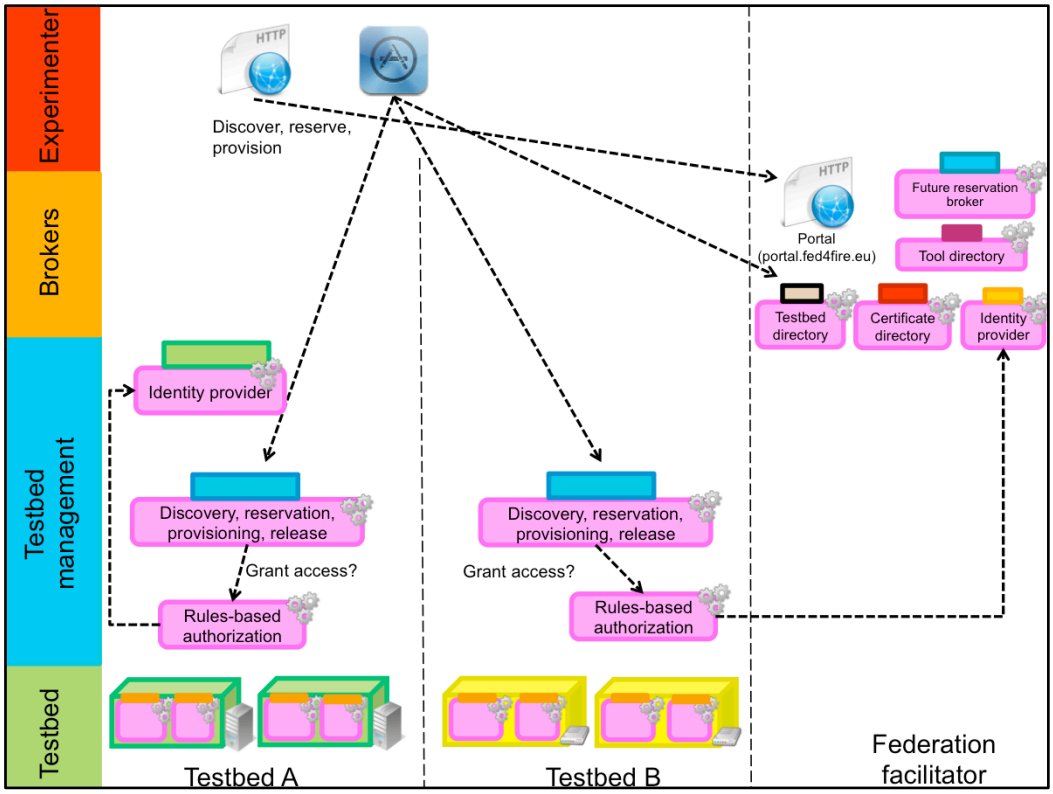
Split up of architecture figures

Figure 1	Figure 2	Figure 3
• Resource discovery • Resource requirements • Resource reservation • Resource provisioning • Resource release	• Experiment control	• Facility monitoring • Infrastructure monitoring • Experiment measuring

FED4FIRE

iMinds
CONNECT.INNOVATE.CREATE


Legend
Server
Embedded PC (x86)
Openflow switch
Openflow ROADM
Virtual Machine
VBox
Ethernet interface
Optical fibre interface
802.11 interface
802.15.4 interface
Bluetooth interface
Software defined radio
3G interface
Software / Service
Interface
Ethernet link
Data path
Online storage
HTTP
Web interface
Command line interface
Standalone client
SSH
SSH client



Here the components of the architecture for cycle 1 of Fed4FIRE are depicted which play a role in the following steps of the experiment lifecycle: resource discovery, resource requirement, resource reservation, resource provisioning and resource release.

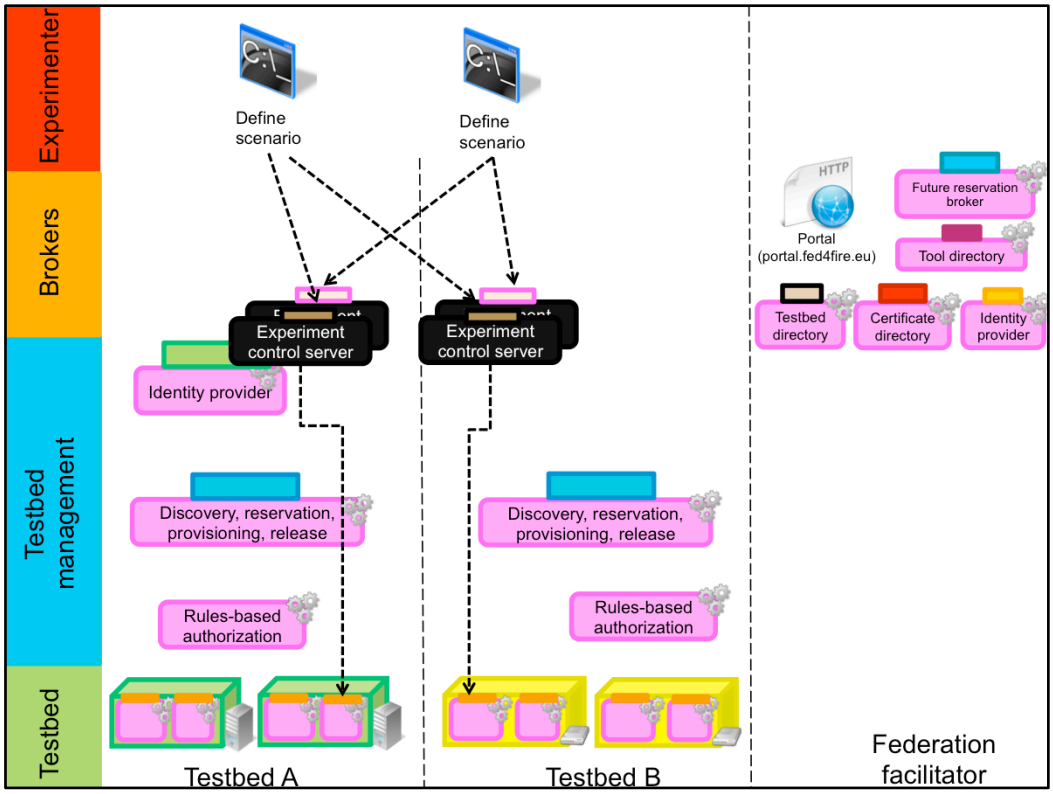
The architecture is distributed with components at the testbed location, some at the federation facilitator (on or more 'central' locations) and the experimenter clients at the experimenter's PCs/laptops/…. It is a goal to add only components in central locations to facilitate the use of the testbeds, but without being necessary for a correct operation. We call these 'brokers', as they provide 'brokered' access between experimenter tools and the testbeds. Compare this to a DNS service in the internet: a browser/application can perfectly use IP addresses to reach services, but DNS eases this by introducing a mapping between a human readable hostname and domain and an IP address. However, DNS is in principle not necessary in the process of reaching the service.

The following components (except the future reservation broker) will be provided at the federation facilitator for development cycle 1 of Fed4FIRE (there are 3 cycles in total, cycle 1 will be operational in February 2014):

- **Portal**: a central starting place and register place for new experimenters
- **Identity provider**: experimenters who register at the portal are registered at this identity provider (as can be seen in the Figure, there will be also identity providers at testbeds)
- A **testbed directory** which is readable by humans and by computers to have an overview of all testbeds in the federation
- A **tool directory** which gives an overview of available tools for the experimenter
- **Certificate directory**: for the chain of trust, there should be a trusted location of root certificates of the identity providers
- **Future reservation broker**: to facilitate future reservations of resources, this broker can help to find the right time slots and resources over multiple testbeds. Instead of an experimenter tool which has to query all testbeds, it can do a single query to this broker.

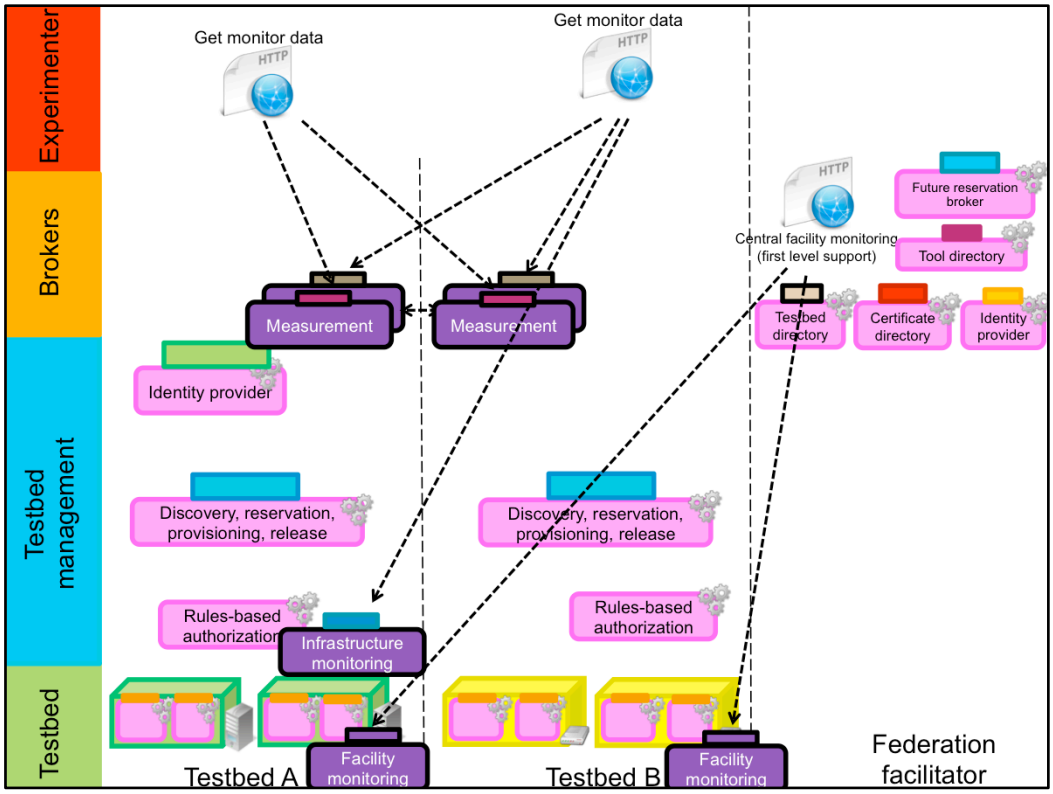
At the testbed side, we have the following components:

- A component which does discovery, reservation, provisioning and release with a **common interface: SFA.**
- A testbed may be or may not be an identity provider
- For authentication/authorization between users and testbeds a trust model is used, where identity providers trust each other and specific experimenter properties are included in the experimenter's X.509 certificate, which is signed by the identity provider. So testbeds can do rules-based authorization.
- A testbed can query/trust the central certificate directory to see which root certificates it should trust.



The starting point for experiment control is that the testbeds or federation facilitator should not run specific experiment control components, as the experimenter can fully roll this out on his own. However the testbed providers could ease this by putting certain frameworks pre-installed in certain images. Figure 3 shows two experiment control frameworks each with their own interfaces and experimenter user tools/command line tools.

In order for testbeds to support these different kinds of experiment control servers, it can be of interest if all such control servers would adopt a common protocol for resource control. In that case, testbeds only have to support this single protocol, and experimenters can deploy any compliant experiment control server/tool in their experiments. Such a common protocols has been recently released: the Federated Resource Control Protocol (FRCP). It is adopted by Fed4FIRE, as is explained on the next slide.



The following types of monitoring and measurement are identified:

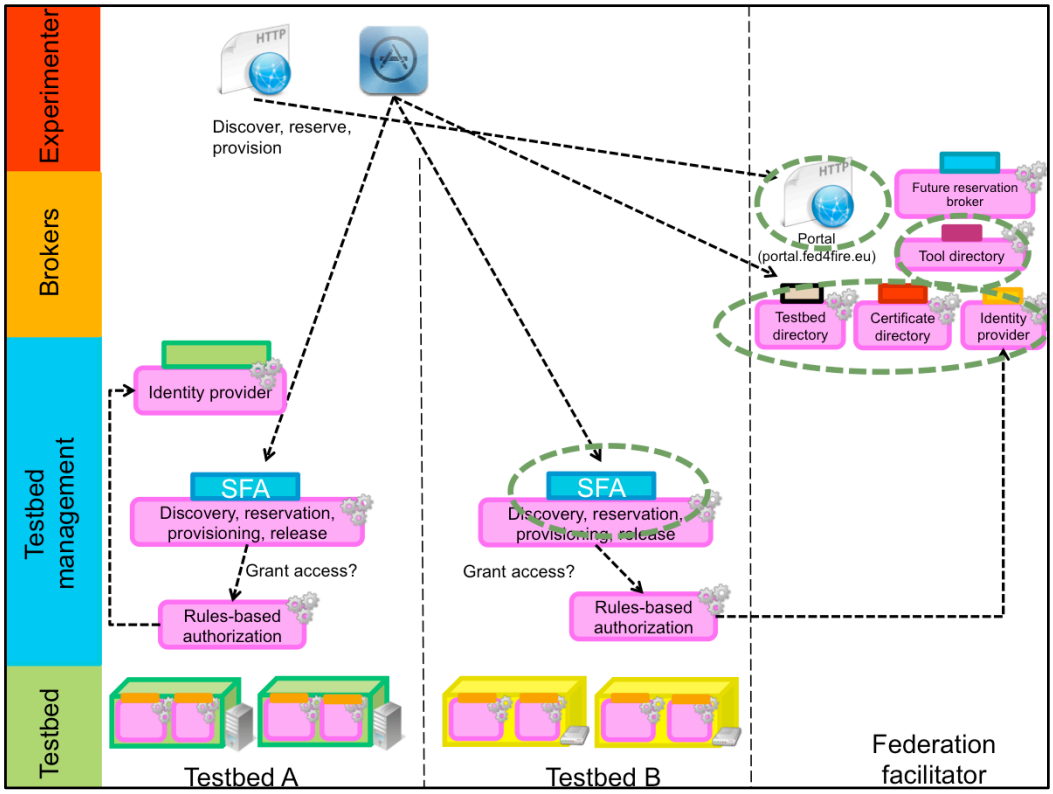
Facility monitoring: this monitoring is used in the first level support to see if the testbed facilities are still up and running. The most straight forward way for this, is that there is a common distributed tool which monitors each facility (Zabbix, Nagios or similar tools). The interface on top of this facility monitoring is the same: OML streams (also see next slide)

Infrastructure monitoring: this is monitoring of the infrastructure which is useful for experimenters but which they cannot do themselves. E.g. monitoring of switch traffic, wireless spectrum or physical host performance if the experimenter uses virtual machines. This should be provided by the testbed provider (an experimenter has e.g. no access to the physical host if he uses virtual machines) and as such a common interface is again applied in the form of OML streams.

Experiment measuring: measurements which are done by a framework that the experimenter uses and which can be deployed by the experimenter itself on his testbed resources in his experiment. In the Figure one can see two experiment measuring frameworks each with its own interfaces (and thus experimenter tools). Of course, a testbed provider can ease this by providing e.g. OS images with certain frameworks pre-deployed. OML is an example of such a measuring framework.

Outline

- What is Fed4FIRE?
- Overview of the Fed4FIRE architecture
- Implementation of the Fed4FIRE architecture
- Implications of joining Fed4FIRE on the testbeds

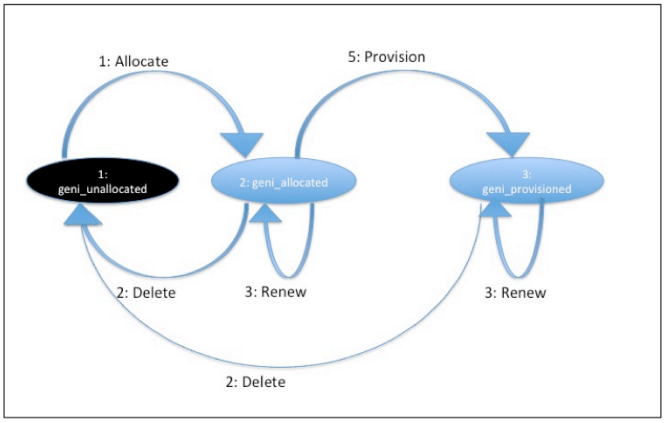


In the following slides we will discuss the implementation details of the following functional components of the architecture:

- Common interface for resource discovery, reservation, provisioning and release: SFA (including Rspecs and the tool SFA Wrap)
- Security related components (all relying on X.509 certificates):
    - Identity provider (deploy at testbed or use the Fed4FIRE one)
    - Certificate directory (procedure for inclusion of a root certificate, trust decision at the testbed)
    - Portal: functionality, interaction with other components

Adoption of SFA

- API: Geni AM v3,
  http://groups.geni.net/geni/wiki/GAPI_AM_API_V3_DETAILS
- XML-RPC over SSL with client authentication using certificates.



Resource state diagram

API methods:
1. GetVersion
2. ListResources
3. Describe
4. Allocate
5. Renew
6. Provision
7. Status
8. PerformOperationalAction
9. Delete
10. Shutdown

FED4FIRE

iMinds CONNECT.INNOVATE.CREATE

(source: http://opensfa.info/doc/opensfa.html)
SFA has been designed to provide a minimal set of functionalities, a *thin waist* if you will, that a testbed can implement in order to enter into a global and interoperable federation. An experimenter in an SFA-based environment can transparently browse resources on any federated testbed, and allocate and reserve those resources.
Because of the potential for a very large number of testbeds, a global federation architecture faces a serious scalability issue. SFA introduces a fully distributed solution in which each peer testbed serves as the authority of reference for the resources that it brings, and each user community, along with its experiments, is represented by an authority (possibly, but not necessarily identified with an individual testbed).
Under the SFA architecture, there is a separation between what is generic and what is testbed-specific. Testbed-specific information is captured in a resource model, called a resource specification (RSpec), which is an XML transported by the SFA layer. SFA itself does not cover such aspects as resource model, policies, reservations or measurements. These functionalities should instead be implemented on top of SFA.

SFA designates a set of four main object types that represent the different entities involved in the testbed federation:
- **Authorities**: these represent testbeds, parts of testbeds to which trust or rights may be delegated, and/or communities of users.
- **Resources:** these consist of nodes, links, or any other experimental resource provided by the testbeds, and exposed to the users.
- **Users:** these are experimenters wanting access to resources.
- **Slices:** a slice is the basic unit of interaction between users and resources. One can think of a slice as corresponding to an experiment, and englobing all of the users and resources associated with that experiment. As its name suggests, slices play a central role in the SFA.

SFA defines a minimal set of API calls to enable interaction between the different actors of the federation, and that are implemented around three main components:
'***Registry manager***': This exposes objects that are managed by the federation. '***Aggregate manager*** *(AM)*': This exposes the resources of an individual testbed, or more generally, the resources that fall under a single authority. ***'Slice manager (SM)'*:** This exposes the resources from multiple, federated authorities and is used to track slice objects.
The API calls listed on the slide can be organized into three main categories:
'Object management': These calls manipulate registry objects through the classical list, create, read, update and delete functions. '*Resource browsing and slice management*': These calls associate resources to slices, as well as starting, stopping or getting the status of slices. *'Federation discovery*': There is an API call that is used to obtain detailed information about the different federation services that are running, and to recursively discover peer platforms.

SFA is based on a web services API. To issue a call, a user must connect to a manager's XML-RPC interface via HTTPS, using their private key as a cypher, and passing as a first parameter the credential that shows that they are authorized to perform the operation.

Adoption of SFA: RSpec

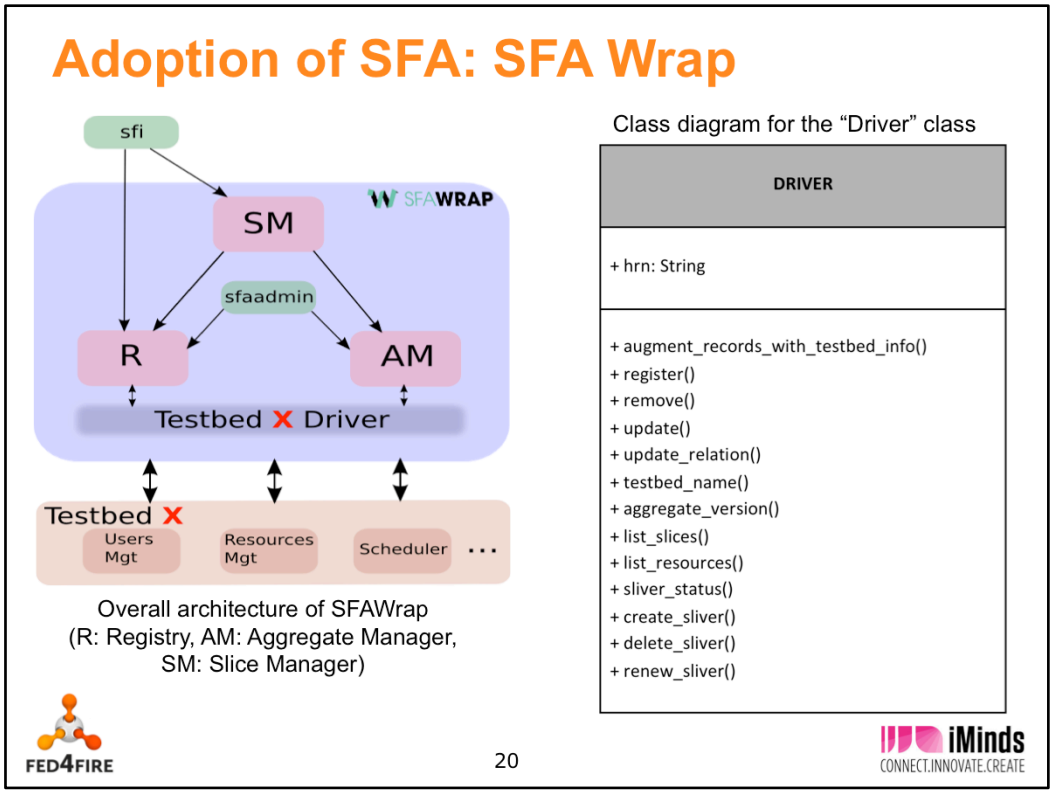
- XML for resource/experiment description
- http://www.geni.net/resources/rspec/3/
- 3 types:
  - Advertisement
  - Request
  - Manifest
- Different domain-specific extensions to the Geni Rspec 3 exist today
- Fed4FIRE is defining a novel ontology-based common Rspec

FED4FIRE

19

iMinds CONNECT.INNOVATE.CREATE

The GENI RSpecs are not tightly specified, which means that the same type of resources (e.g. virtual machines) are defined in multiple ways. It is the goal in Fed4FIRE to explore the use of ontology based descriptions for these RSpecs. This should make it more easy for experimenters, experimenter tools and broker developers to use these resources.

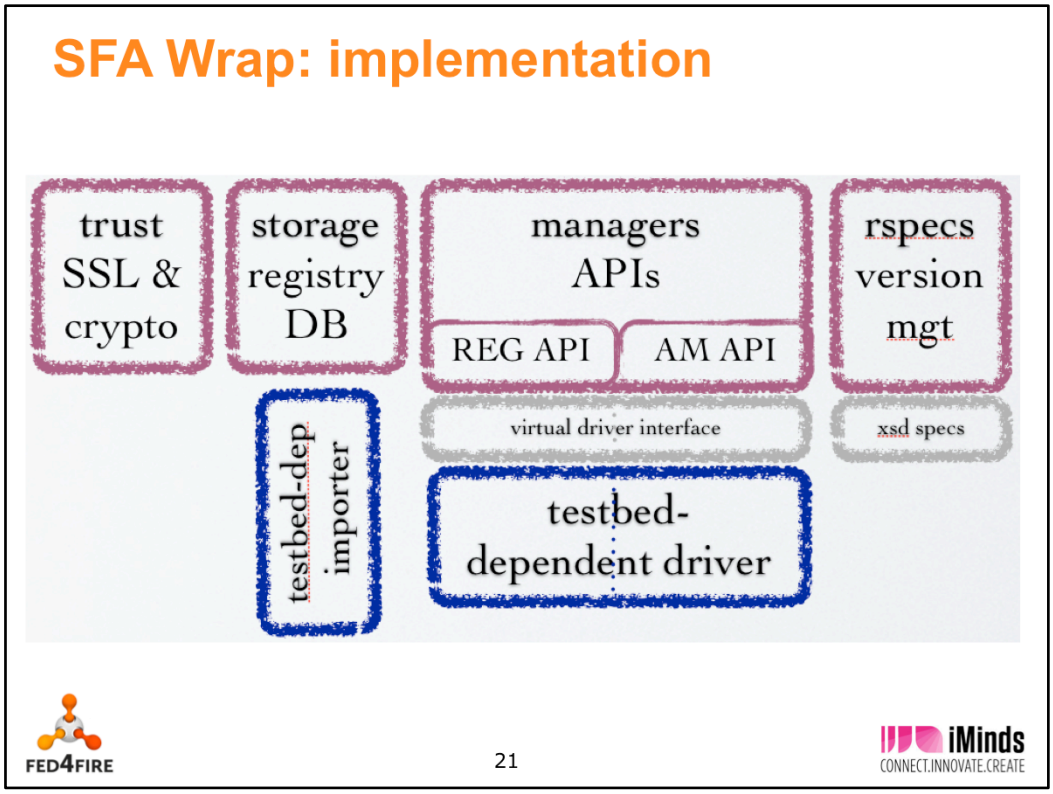


The main idea in the Fed4FIRE SFA approach is that we leave testbeds the freedom to choose how they want to implement this SFA interface on top of their testbed management systems. However, for testbeds that have not yet dug deeper into the SFA route, the project provides a generic tool that allows them to adopt SFA with limited efforts and in a relatively short amount of time. This tool is called SFA Wrap. It can be described as the generalized implementation of the SFA interface on top of PlanetLab Europe. This SFA Wrap 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. 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 slide depicts the main components of the SFAWrap overall architecture.

- **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.

The slide also depicts the UML class diagram of the 'driver' class. 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.



Initially SFAWrap code was targeting PlanetLab testbeds only. With this in mind, the whole code was redesigned to reach the design that is depicted on the slide. 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.

In more details, the various parts of the generic code are:

- **trust:** This package implements all the 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. 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.

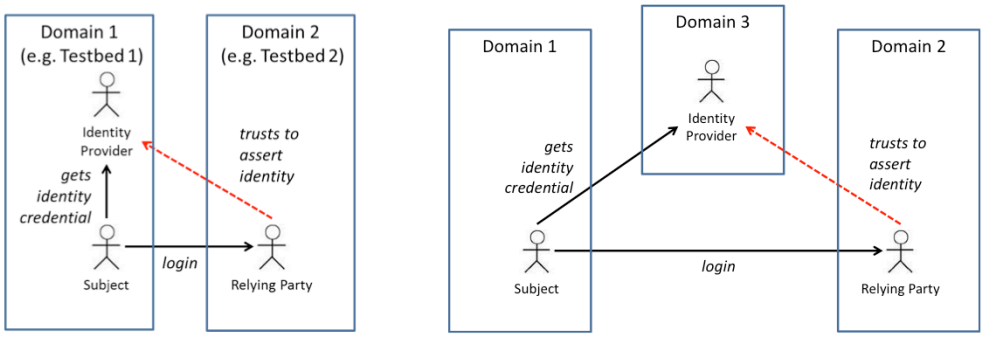
More hands-on information about how to use the system from a programmer's point of view can be found at https://svn.planet-lab.org/wiki/SFADeveloperTutorial

Security

- Based on X.509 certificates
- This impacts the following components:
  - Identity provider
  - Certificate directory
  - Rules-based authorisation (not part of cycle 1, and hence not relevant for facility providers at the moment)

Security: identity provider

- Testbeds can run their own identity provider, providing SFA-compliant X.509 certificates to their own users
- Testbeds can also choose to outsource this functionality to the central Fed4FIRE identity provider



Testbed 1 runs its own identity provider

Testbed 1 relies on the central Fed4FIRE identity provider

Testbeds have two different options to give their local users an identity that is valid throughout the Fed4FIRE federation: they can run their own compliant identity provider, or they can request their users to register with the Fed4FIRE central identity provider. Given that there may therefore be multiple identity providers in the federation, there is no requirement for the implementation at each site to be the same technology. As long as the technology is capable of producing X.509 certificates in the right format and is robust, mature and well supported, it may be considered an implementation candidate.

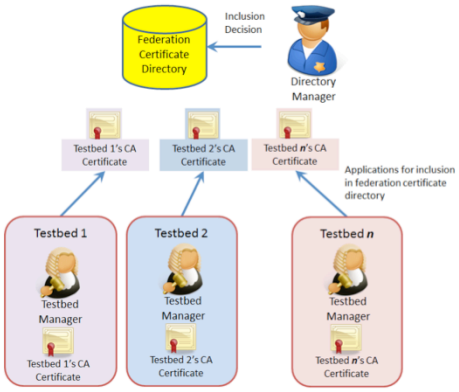
In order to create a key pair and X.509 certificate, the following steps are required:

1. The user must create a public / private key pair. The public / private key pair is owned by the user, and the private key should never go out of the possession of its owner.
2. The user must create a “Certificate Signing Request” (CSR) that binds the public key to some identity information.
3. The user must submit the CSR to the Certification Authority (CA) for signature. This may be a CA created by the testbed or an external CA.
4. The CA performs any checks required, and if satisfactory, it signs the CSR with its CA certificate. The result of this is the user’s certificate, which binds the user’s identity information with the user’s public key, and the whole bundle is signed by the CA certificate.
5. The CA returns the user’s certificate to the user.

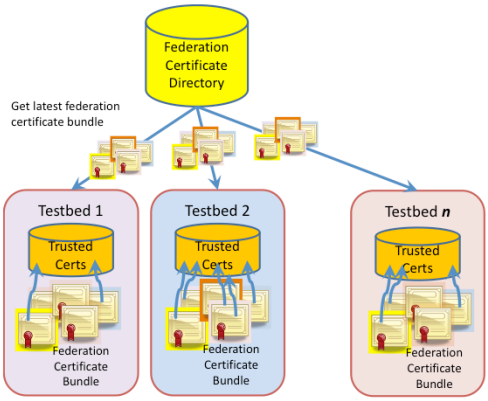
Note that the SFAWrap Registry component can be used as an SFA-compliant X.509 identity provider.

Security: certificate directory

- Testbeds will only grant access to experimenters belonging to trusted identity providers.
- To authenticate the relation between an experimenter and its identity provider, testbeds need to have the root certificate of that provider.
- The certificate directory is a federation-wide store of root certificates that can be consulted by users and testbed providers.



Procedure for inclusion of a testbed's certificate in the directory



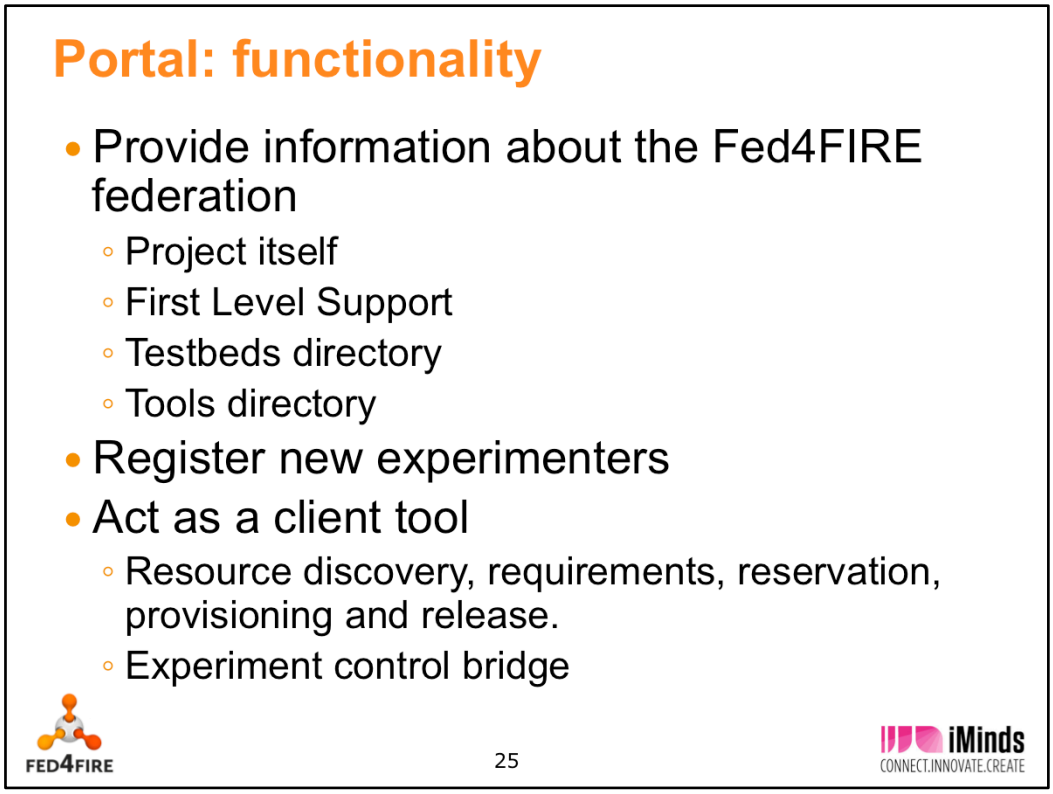
Trust decision at the testbed using the certificate directory

The certificate directory is a federation-wide store of root certificates that can be consulted by users and testbed providers. In many cases, testbed providers act as their own Certificate Authority (CA) by having a self-signed certificate that is used to sign other certificates for users of the testbed. The certificate directory is a single place to store all testbed providers' "CA" certificates, for use by other federation participants. In use, the certificate directory is likely to be consulted on a regular basis by (for example) testbed providers, to provide them with an up to date list of the "CA" certificates used by other testbed providers to issue their certificates. The certificate directory is simply a single place to put CA certificates - the final decision whether to trust a certificate in the store is down to the individual user or testbed.

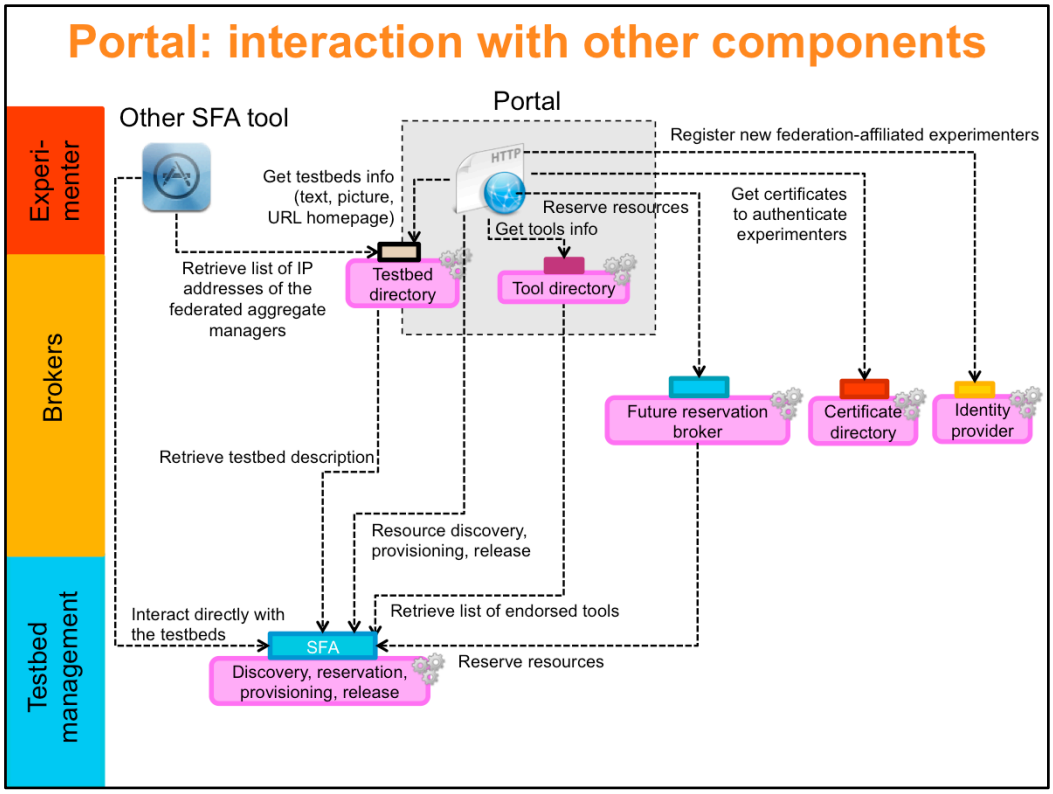
The certificate directory is also an ideal place to store revocation lists from all the CAs - these are blacklists of certificates a CA has issued in the past but now wishes to revoke. Revocation lists are used by parties who check signatures (in this case the most obvious party is the testbed) because they need to know the certificates that are blacklisted, and therefore they need to regularly download the latest revocation lists.

There are access regulations in place as to who can put items (certificates or revocation lists) into this store, as it must not be possible for anyone to put an item in the store (or for that matter to delete an item). Therefore, for "write" access to the store, there are special privileges given to a trusted person who manages the certificate directory. The procedure for getting an item into the certificate directory should be that an application is made to the certificate directory manager (as shown on the slide). The directory manager can use some criteria to determine acceptability of the requester, and if accepted, the manager puts the item in the directory using its privileged access rights.

Getting a certificate or revocation list out of the store is simply a matter of it being served to the requester. The adopted technology for this download is HTTP. Since certificates and revocation lists are files in their own right, the certificate required can be identified by name. Alternatively, a bundle of all the certificates or revocation lists can be downloaded from a single request. It is important to note that it is the testbeds' decision which CA certificates they decide to trust - simply by having a certificate in the federation certificate directory does not imply any obligation onto the testbed providers to trust it. This is illustrated in the right figure on the slide. Each testbed can download the complete bundle of CA certificates for the federation, but the testbed has the final decision which of these certificates they trust. In the figure, this decision is denoted by the arrows indicting that a testbed has put the certificates they trust into their own local trust store.



The portal is the central starting point to access the Fed4FIRE federation. The portal provides pointers to the project website, to first level support (WP8), to the federated testbeds' background information and to appropriate FIRE tools. Portal users are also 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. The portal also acts as the registration place for new experimenters, providing an easy way for experimenters to register themselves with the Fed4FIRE identity provider (and to access the federated testbeds). Note however that the testbeds always determine whether the user can actually access them according to their access policies. Moreover, the portal also performs the role of a client tool. On behalf of the experimenter, the portal forwards queries to federated testbeds using SFA delegation mechanism. Using the portal, the experimenter is able to search, browse and reserve resources across federated testbeds. The portal also acts as a bridge to experiment control tools.

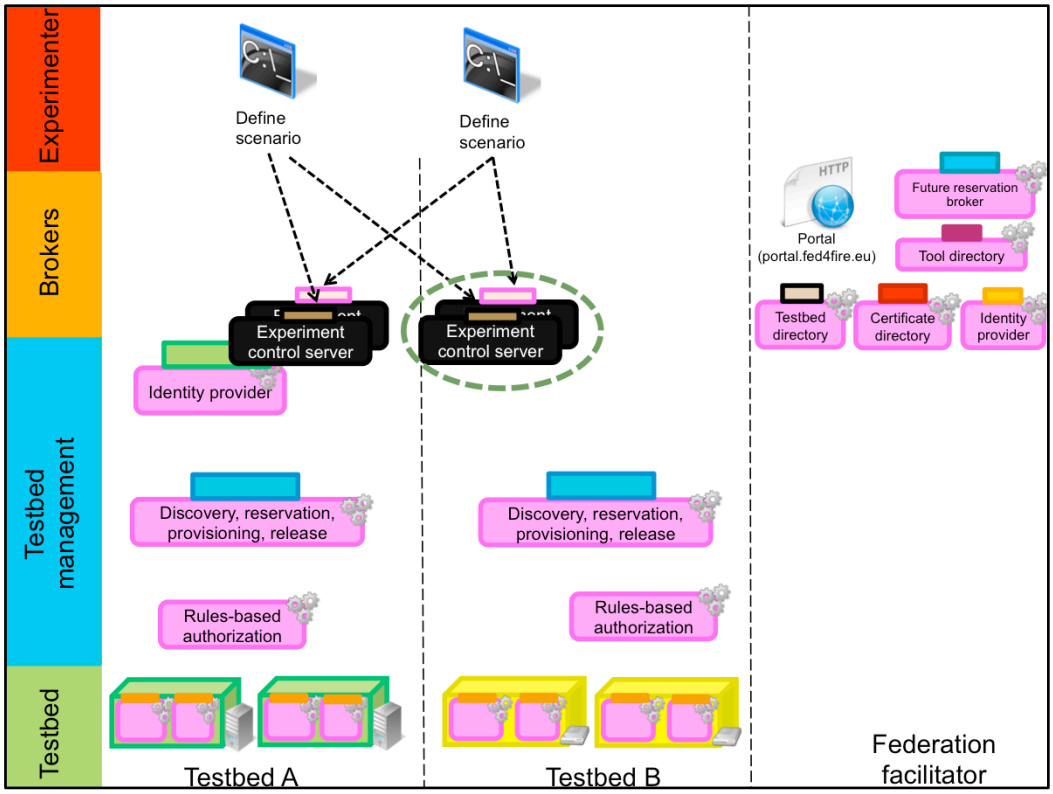


It is obvious that the portal provides a vast amount of functionalities to the experimenters. Its implementation is composed of three different functional elements of the Fed4FIRE architecture:

- The **portal web interface**. This is an instance of the MySlice framework. It is open to anyone with a valid Fed4FIRE identity (being registered with the central Fed4FIRE identity provider or any of the testbed-operated Fed4FIRE compliant identity providers). This web interface implements the functionalities related to experimenter registration and to acting as a client tool. The framework can be easily extended through the development of MySlice plugins.
- The **testbed directory** is a central service that provides the pointers to the different testbeds belonging to the Fed4FIRE federation. It implements two different access methods: human readable and machine readable. 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, a picture, and a URL pointing to a particular testbed homepage where more detailed information is provided.
- The **tools directory** provides pointers to FIRE tools of all kinds, both to the officially endorsed ones and to those tools that emerged naturally from the FIRE community. A mechanism using the SFA GetVersion API call is in place in order to enable testbed providers to indicate the tools that they officially endorse on their testbed.

The interactions between these three components and the other components needed for their successful operation are indicated on the slide. The portal web interface retrieves the needed information from the testbed- and tools directory and presents this to the experimenter. Both directory components will retrieve their needed information directly from the different testbeds using the SFA interface, based on a small extension of the value returned by the GetVersion SFA API call that was defined by Fed4FIRE. The same testbeds' SFA interface is also contacted by the portal's web interface for resource discovery, provisioning and release. For user authentication, the portal web interface will rely on the root certificates downloaded from the certificate directory. Using the portal, users can also register themselves with the central Fed4FIRE identity provider when needed. The portal web interface also provides a GUI to reserve resources within the federation. For this it interacts with the future reservation broker, which in its turn interacts with the different corresponding testbeds to perform the actual reservation.

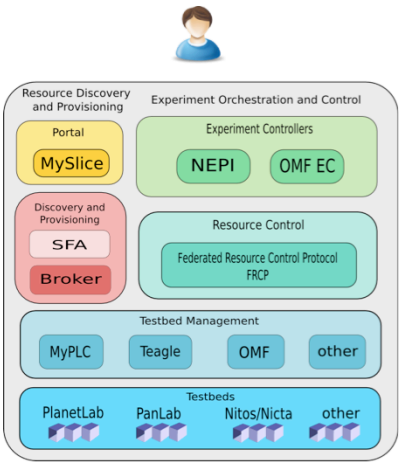
Note that next to the portal, any other SFA experimenter tool could be used to provide similar functionalities, in line with our architectural approach that there should be no single point of failure in the federation. As depicted on the slide, such a SFA tool could get the lists of the IP addresses of the Fed4FIRE testbed AM's, and then directly interact with the testbeds' SFA interfaces to implement the same functionalities as provided by the portal. Note that it could (and most likely should) also interact with the Future reservation broker, certificate directory and identity provider. This has been omitted from the figure in to avoid overloading it. Also be aware that in this approach the testbed directory is not considered as a single point of failure, since the experimenter would also be able to retrieve the IP addresses manually from other sources, and import them in the SFA tool.



In the following slides we will discuss the main implementation detail regarding experiment control: the adoption of the FRCP protocol as a common resource control layer. Since this protocol is closely related to the latest OMF6 developments, we also briefly introduce the OMF architecture.

Adoption of FRCP

- Experiment control
  - Resource control mechanism
  - Experiment controller
- FRCP:
  - Federated Resource Control Protocol
  - Common protocol for RC
  - Efficient way to support RC in heterogeneous federation:
    - Multiple testbeds
    - Multiple experiment controllers



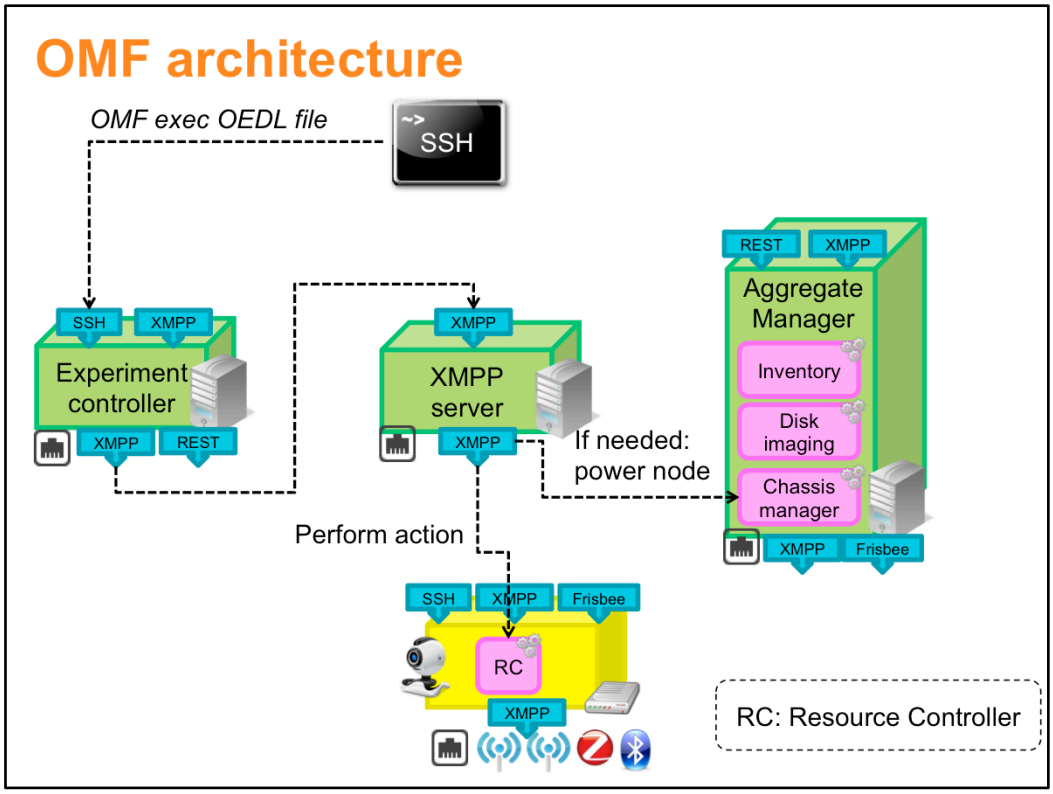
FED4FIRE

28

iMinds CONNECT.INNOVATE.CREATE

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.

Regarding the control layer, 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. 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. This technology is currently driven in the context of OMF 6 development, defining the messages characteristics for the FRCP, and several resource controllers suitable for a wide range of testbed. Fed4FIRE adopts FRCP as its common experiment control layer. Therefore it needs to be supported by all Fed4FIRE testbeds. As a result, all experiment controller engines compatible with FRCP will be able to access and control resources provided by federated facilities out of the box. Both NEPI and the OMF Experiment Controller are supported in the Fed4FIRE federation, the choice of using the one or the other will be related to the type of experiment. But in theory any other FRCP compliant controller should also be able to work out of the box.



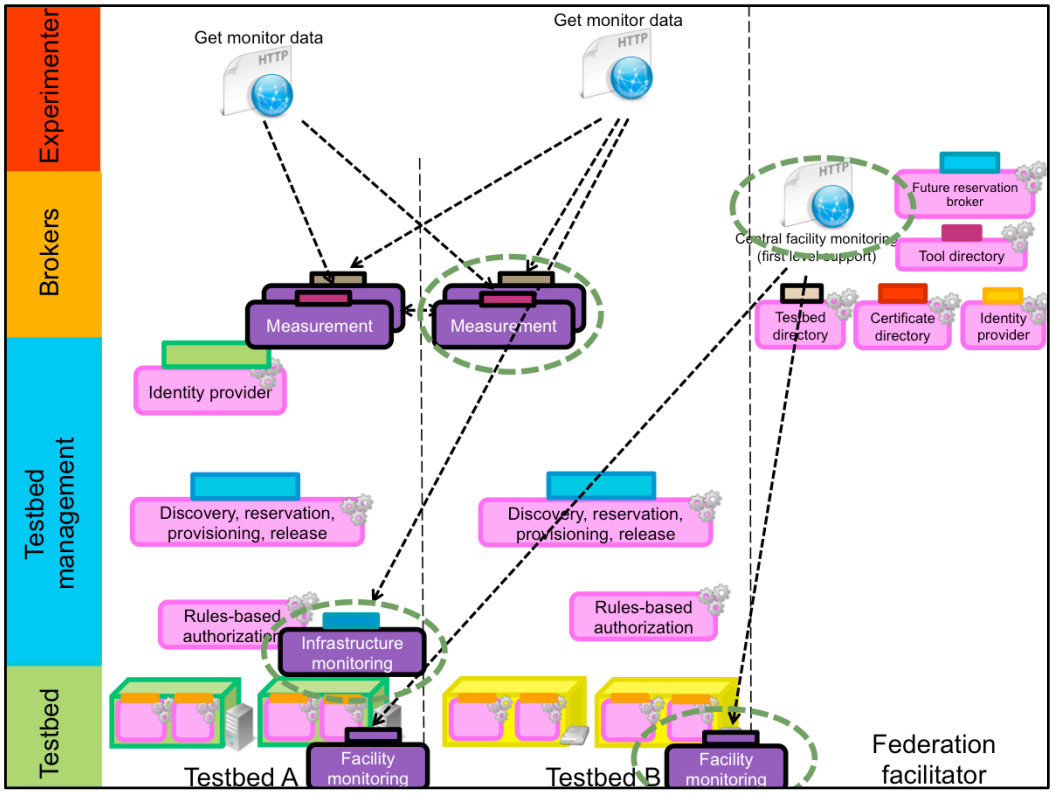
FRCP is currently under development as part of OMF, introduced for the first time in OMF6. Therefore the easiest way for testbeds to adopt FRCP will be to install the FRCP related part of OMF6. Therefore it is of interest to briefly introduce OMF here in some more technical detail. OMF (http://omf.mytestbed.net) 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 (OEDL). OMF also defines a generic resource model and concise interaction protocol (FRCP), which allows third parties to contribute new resources as well as develop new tools and mechanisms to control an experiment (as mentioned on the previous slide). It has a distributed communication infrastructure based on XMPP supporting the scalable orchestration of thousands of distributed and potentially disconnected resources.

But how does this OMF experiment control work? This is depicted on the slide. At the bottom the resources are depicted, in this case an embedded PC with several wired and wireless connections. On the OMF layer, three entities can be observed. The **Aggregate Manager (AM)** is responsible for the inventory, disk imaging and chassis management (powering nodes on or off when needed. The second OMF management entity is the **Experiment Controller (EC)**. It processes a description of the experiment scenario (written in the OEDL language), and will automatically trigger the right actions at the right nodes when needed. Although it is part of the OMF management layer from a logical point of view, the EC can both be provided as a service by the testbed, and can be run locally by the experimenter on its own PC. To perform these actions at the resource, the EC will send a message to a daemon running an every resource: the **Resource Controller (RC).** The RC is capable of executing everything what a user could do manually on the command line. It can also make abstraction of certain commands by wrapping them in OMF commands. An example is the OMF command to change the Wi-Fi channel. Behind the curtains it will determine which wireless driver is used on the resource, and will then select the suitable set of command line commands to execute, depending on the driver. To support this messaging between EC and RC, an XMPP service is used. Hence a **XMPP server** is added as the third entity of the OMF control framework. The protocol used by the EC to request actions at the different RCs is FRCP.

So the experiment control is executed as follows:

1. The experimenter gains access to the PC that runs the EC (his own PC, or a server at the testbed that is reachable through SSH)
2. The experimenter starts the experiment control procedure by giving the command “OMF exec”. The name of the scenario description that is to be executed is given as an argument. The EC will process this description, and will initiate specific commands at certain resources at the appropriate time.
3. If the target node is powered off, the EC will call the chassis manager service of the AM to power it on. For this it will send an appropriate XMPP message.
4. Once the target node is powered on, the EC will request the RC running at the desired resource to perform a certain command. As mentioned, this can be any command that could also be given on the command line manually. To trigger the RC, an XMPP message is sent from the EC to the RC. This message is compliant with the FRCP protocol.

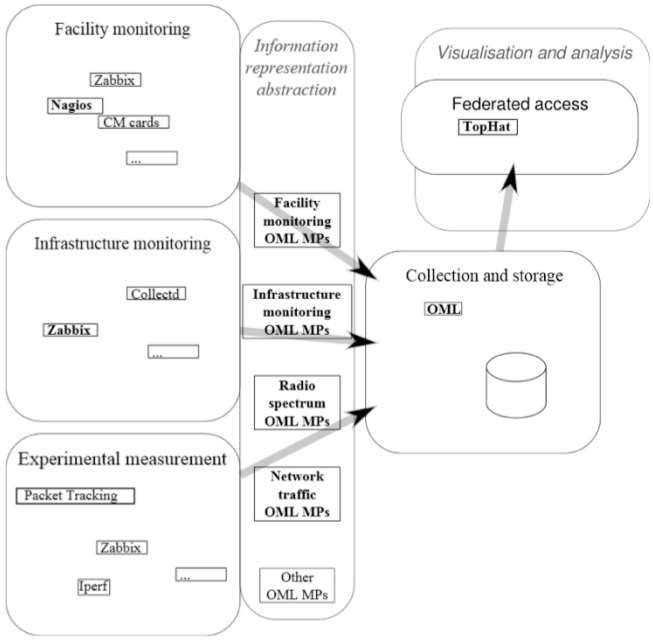
It is a common misassumption that OMF and OML are the same things, while in fact they are not. OMF is the framework for provisioning and experiment control as described above. OML is an additional software suite targeting experiment monitoring. They are very often deployed together in testbeds, but this is not a strict requirement. From a logical point of view they can be considered as two separate entities. You can run OMF without running OML, and vice versa.



In the following slides we will discuss the main implementation detail regarding monitoring and measuring: the adoption of the OML stream as a common data format. The measurement, infrastructure monitoring and facility monitoring will output OML streams, which can then be accessed, persisted and visualized using experimenter client tools.

Adoption of OML stream as common format

- Uniform reporting of data through OML streams
- Freedom of choice regarding M&M tools
- Collection and storage: OML framework
- Visualisation and analysis: Tophat

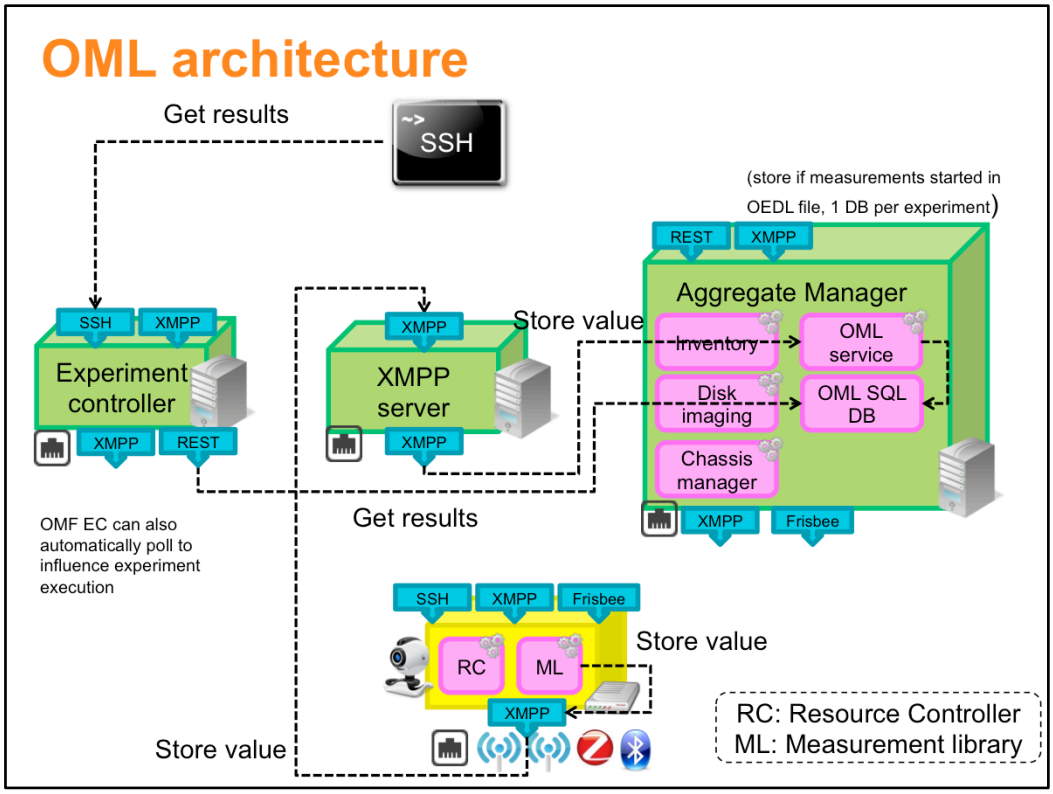


Thorough analysis led to the insight that the most widespread commonality is the use of OML as a collection and reporting framework. It allows instrumenting any sort of system through the abstraction of a measurement point, describing a group of metrics. This abstraction allows more latitude in the choice of measurement tools: as long as they conform to the same measurement point for the same information, their output can be used indistinctly in further analysis. Selecting OML for reporting purposes therefore allows flexibility in the choice of measurement tools, both for monitoring and measurement tasks, as well as for a unified way to access the collected data.

This however only caters for collection and storage, but not directly access to or visualization of the data, let alone from a federated environment. Another tool for this purpose therefore needs to be identified. For this purpose, Top Hat fits the bill for its ability to run queries over distributed systems and data stores, and pre-existing deployments

Facility and infrastructure monitoring tasks require specific metrics to always be made available about the testbed and its node. While some deployments already have solutions in place, the most indicated ones for others are, in order of preference, Zabbix, Nagios or Collectd.

Overall, this caters for the measurement architecture shown on the slide. Essentially, all testbeds will be required to deploy OML and TopHat, for measurement collection and federated access, respectively. With the aim of limiting the impact on deployed solutions, monitoring and measurement tools already in use will not be superseded, but rather adapted to be included in the proposed architecture. For cases where a requirement is not met, default solutions are prescribed.

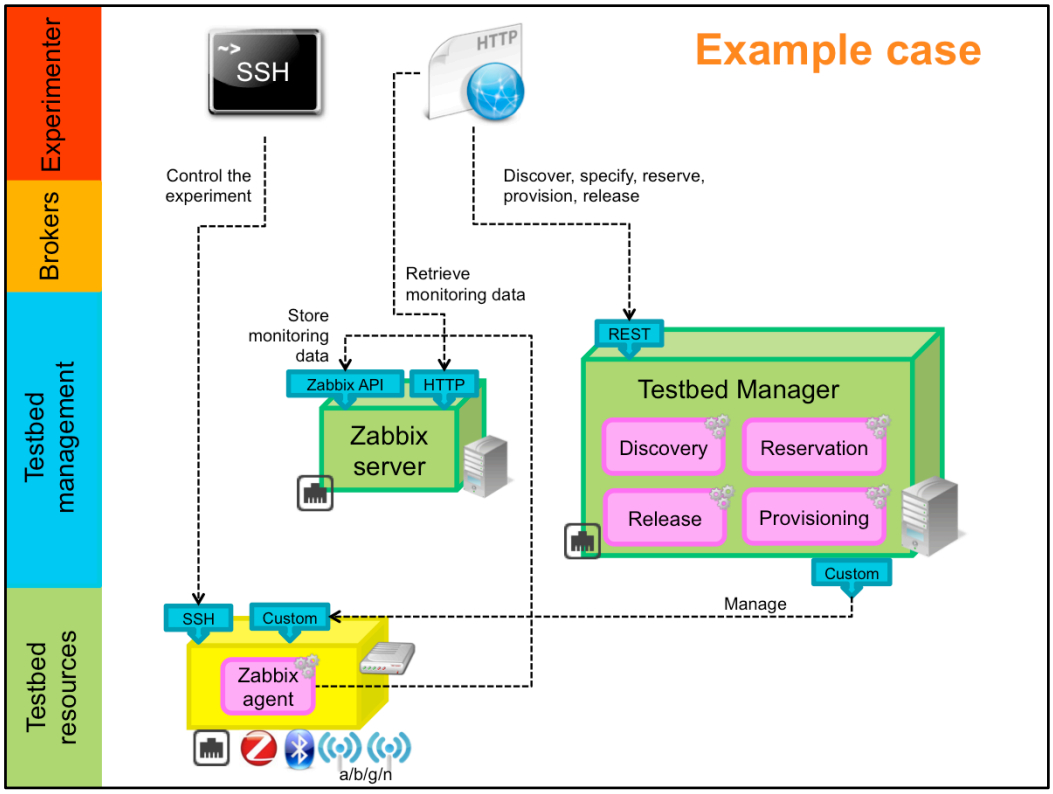


OML (http://oml.mytestbed.net/projects/oml/wiki) is a generic framework that can instrument the whole software stack, and take input from any sensor with a software interface. It has no preconception on the type of software to be instrumented, nor does it force a specific data schema. Rather, it defines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

As mentioned before, OMF and OML are separate entities from a logical point of view: you can run OMF without running OML, and vice versa. But their corresponding software libraries are installed on the same entities. As depicted on the slide, OML consists of a service running on the resource, and a service and database running on the Aggregate Manager (AM). On the resource, the Measurement Library (ML) takes measured values as an input, and is responsible for getting them added to the database at the AM. It does so by calling the OML service running at the AM. Again XMPP messaging is applied. Annotations to the measured values such are experiment ID, source ID and so on are automatically added by the OML framework. From an experimenter point of view, it is sufficient to redirect the measured value coming out of your own software or measurement tool to the ML to collect all of them in a single place for future processing.

Outline

- What is Fed4FIRE?
- Overview of the Fed4FIRE architecture
- Implementation of the Fed4FIRE architecture
- Implications of joining Fed4FIRE on the testbeds



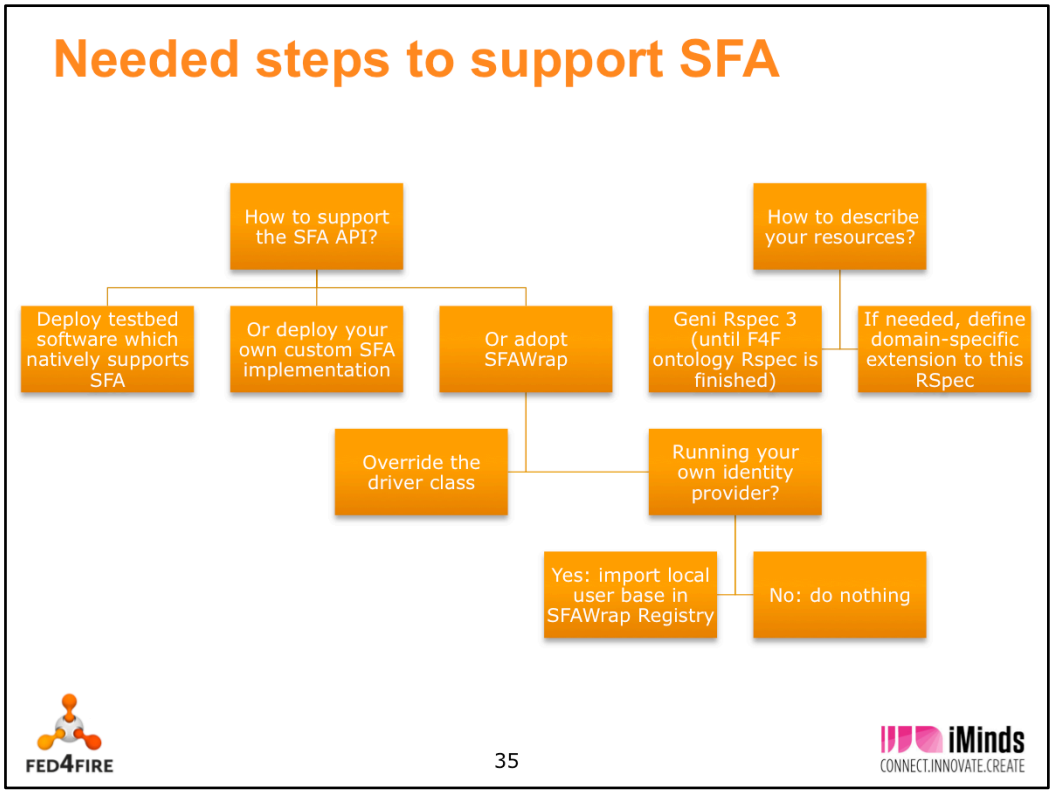
In order to illustrate the implications of joining Fed4FIRE on a testbed, we will ellaborate an example case. This example is a wireless testbed which has the following type of resources: embedded PC's with several wireless interfaces: IEEE 802.15.4/ZigBee , Bluetooth and 2 IEEE 802.11 a/b/g/n interfaces.

This testbed is managed by specific testbed management software. It is responisible for providing resource discovery, reservation, provisioning and release services to the testbed's users. These experimenters user their browser to surf to a webpage offered by the testbed manager in order to use these services. The interfacing between the testbed manager and the resources that it manages is a custom one.

Using their SSH client, experimenters can log in to their provisioned nodes, and manually control the experiment on the command line.

To provide the experimenters with monitoring capabilities, the testbed is also equipped with a Zabbix server. The corresponding agents are deployed on every resource. These agents continuously monitor a large number of metrics such as CPU load, free RAM memory, amount of transmitted packets for each networking interface, amount of transmission errors for each networking interface, etc. The corresponding data is pushed to the Zabbix server using the Zabbix API, where it is stored in a database.

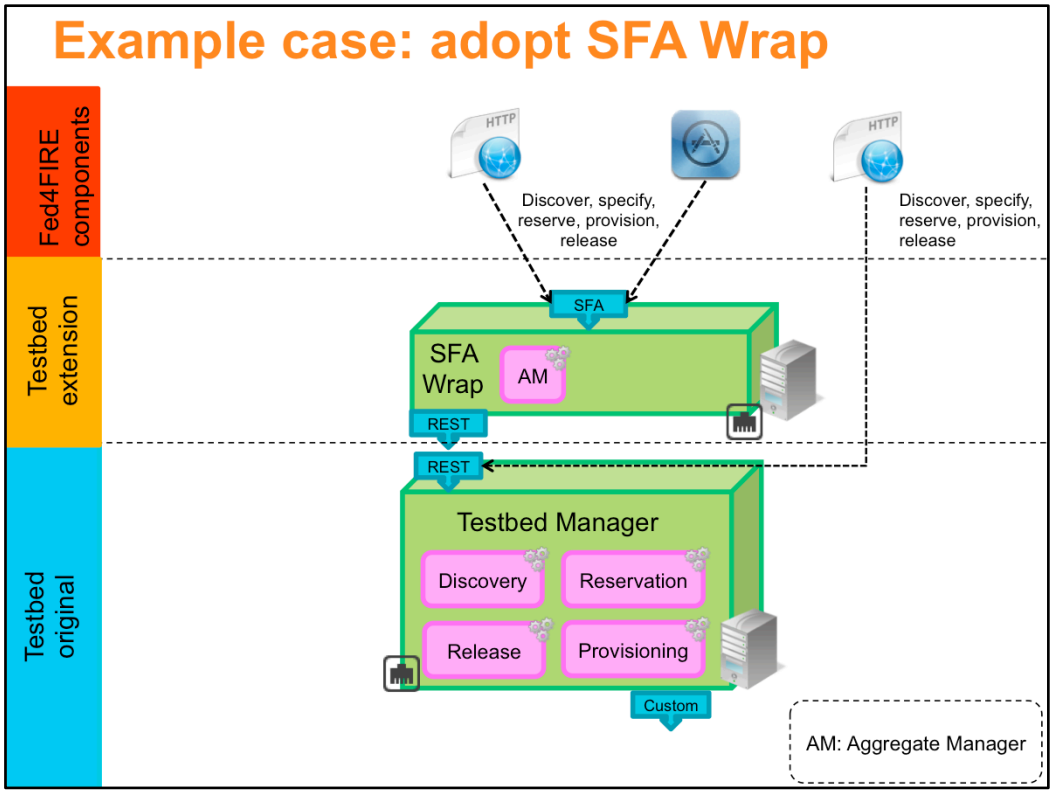
To retrieve this data, the experimenters user their browser to surf to a website provided by the Zabbix server.



When looking to support SFA, a testbed has three different options:

- Replace your existing testbed management software with software that natively supports SFA
- Deploy your own custom SFA implementation that complements your existing testbed management software, or
- Adopt SFAWrap to complement your existing testbed management software. In this case:
    - Override the driver class of SFAWrap
    - When running your own identity provider: import local user base in SFAWrap Registry

Until the common ontology-based Rspec gets finalized: use Geni Rspec 3 Rspecs, define domain-specific extensions of the Geni Rspec 3 if needed

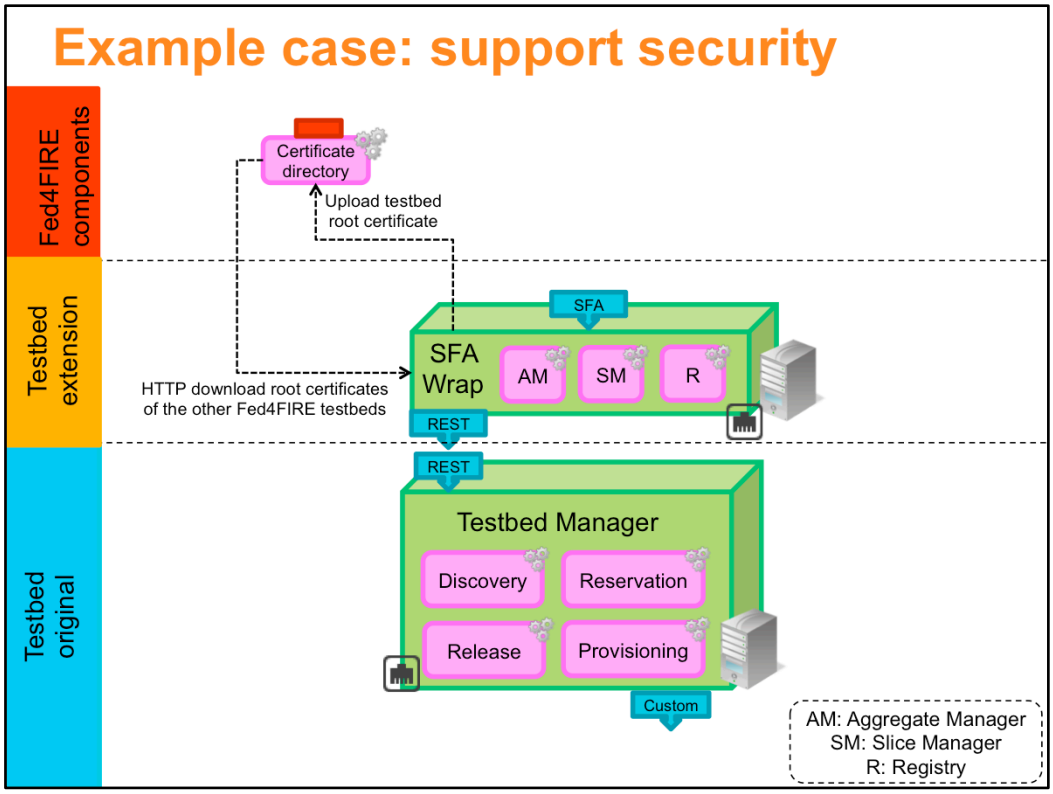


In this example case it was decided to adopt SFA Wrap to support SFA. It is deployed as an additional machine belonging to the testbed management infrastructure, and it translates the incoming SFA AM calls to appropriate calls to the REST interface of the Testbed Manager. For this it was needed to implement the appropriate SFA Wrap driver.

After deploying SFA Wrap and its AM service component, any SFA compliant tool (of course including the Fed4FIRE portal) can be used for resource discovery, specification, reservation, provisioning and release. Of course, the original testbed webinterface also remains operational for serving e.g. local experimenters that prefer to use this interface.

Needed steps to support security

Functional element	Implementation strategy
Identity provider	• Outsource this functionality to the Fed4FIRE identity provider, or • Deploy your own, using the SFAWrap Registry or any other implementation of your choice.
Certificate directory	• If you run your own identity provider: provide its root certificate to the operator of the certificate directory • Fetch the certificates from the certificate directory (using a HTTP get request)



Supporting the SFA AM API is the minimal requirement for testbeds wanting to federate with Fed4FIRE. However, the example testbed's architects decided in this example that they wish to also operate their own identity provider, and their own slice manager. By supporting all this functionality themselves, they opt to rely as little as possible on the services provided for convenience by the Fed4FIRE Federation Facilitator. Therefore the Slice Manager (SM) and Registry (R) services of SFA Wrap are also activated. Since the testbed management software did not yet implement the concept of local user accounts (the entire testbed was operated behind a firewall, once you gained access to the LAN after the firewall it was assumed that you are a local user and can use the testbed freely), there is no need to synchronize the SFA Wrap Registry with any of the testbed management components. Note that the SFA interface should of course be exposed directly to the Internet, while the outgoing REST interface used by the testbed-specific driver should be able to then penetrate the firewall.

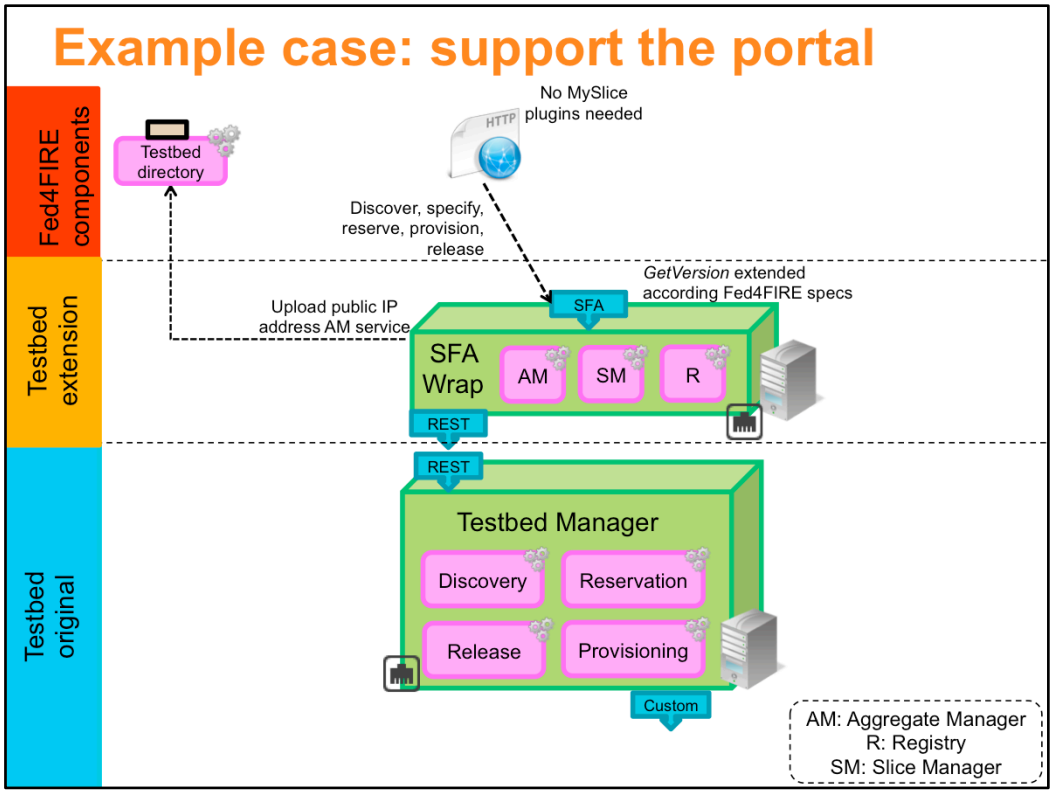
Since the testbed decided to operate its own identity provider, it has to provide its own root certificate to the operator of the Fed4FIRE certificate directory. To be able to authenticate non-local Fed4FIRE experimenters it is also needed to download the root certificates of the other Fed4FIRE testbeds from the F4F certificate directory, and to add them as trusted certificates in the SFA Wrap deployment.

Needed steps to support the portal

Functional element	Implementation strategy
Portal plugin	• Expose your testbed through SFA • The portal provides a table view and a map view on the resources. If your testbed requires a specific GUI, implement this as a MySlice plugin
Testbed directory	• Expose your testbed through SFA • Provide the public IP address of your SFA Aggregate Manager • Extend the XML-struct that is returned by the SFA *GetVersion* call with some high level information about your testbed.
Tool directory	• Support the testbed directory • Augment the high level testbed information with a list of officially endorsed tools

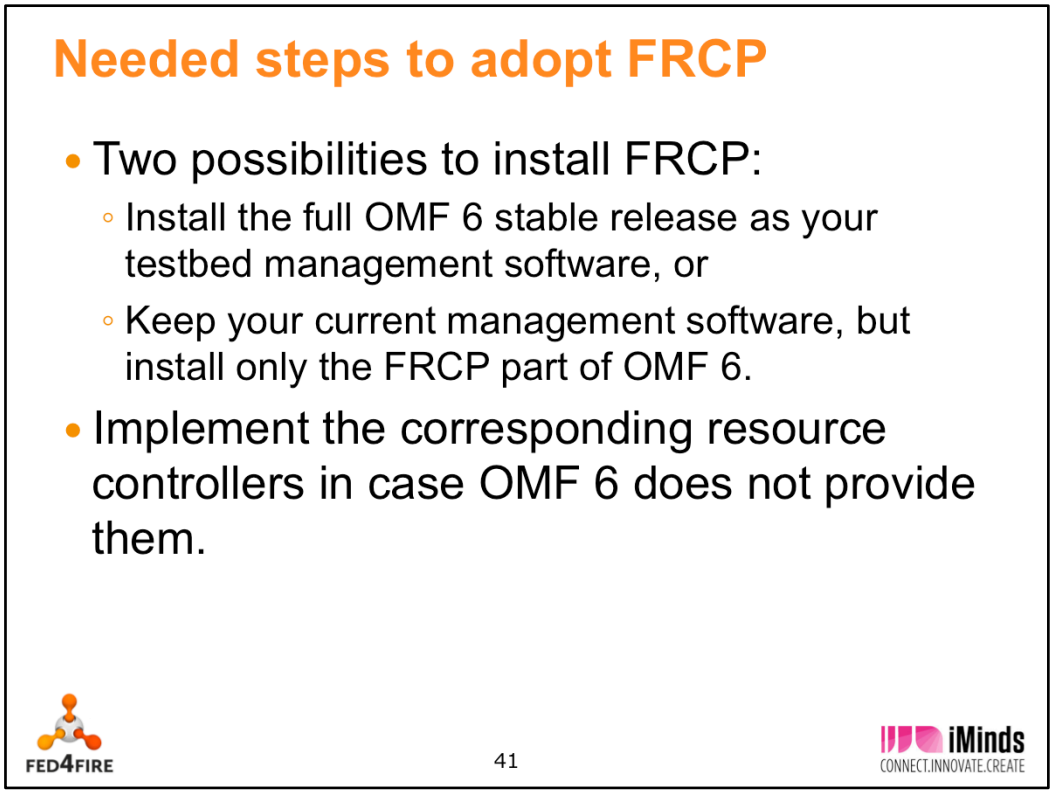
To support the Fed4FIRE testbed and tools directories, the value XML-RPC struct returned by the *GetVersion* API call of SFA is extended with the following members:

- *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.
- f*4f_testbed_homepag*e: 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.



The requirement that the testbed should be exposed is already covered by the fact that we adopted SFA Wrap previously. The existing table and map views provided by the portal are considered to be sufficient. Hence no additional plug-ins for the portal web interface need to be implemented. The public IP address of the SFA Aggregate Manager is communicated to the testbed directory operator, as depicted on the figure. Finally, the GetVersion API call implementation in SFA Wrap is extended in order to include the information needed by the testbed directory and the tools directory.

So no new components had to be added to support the portal, only the public IP address had to be given to the testbed directory operator, and the GetVersion reponse implementation on the deployed SFA Wrap instance needed a minor update.



Some more technical details:

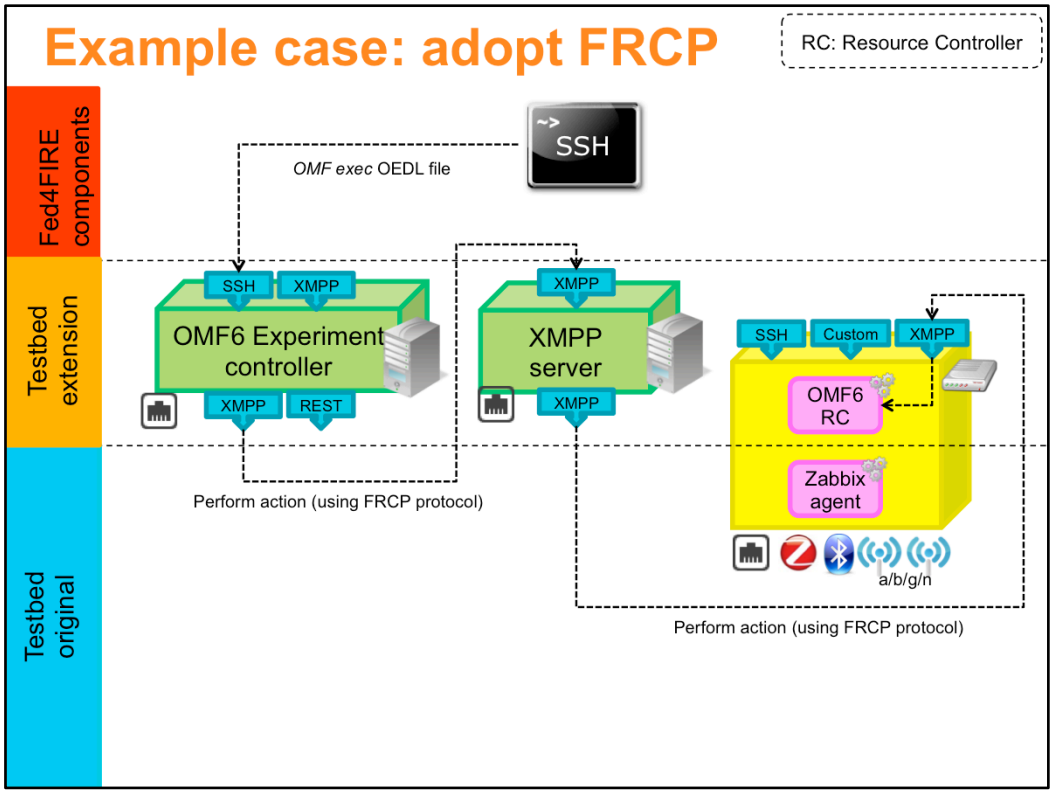
To install OMF 6 in testbeds these are the dependencies the testbed should support:

- 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.



In this example, it is chosen not to fully adopt the OMF6 framework (being its capabilities both for testbed management as for experiment control), but instead to keep the existing testbed management software and install only the experiment control related FRCP part of OMF6 on top. This means that on every resource an additional service is deployed: the **OMF6 Resource Controller (RC)**. Since our resources are all Intel x86 compatible nodes running Linux, there was no need to implement a new OMF6 RC, the appropriate one is already existing as part of the standard OMF6 framework.

It is not a strict requirement that the testbeds deploy an **OMF6 Experiment Controller (EC)** themselves, since experimenters can also run this themselves. However, since it makes life of the experimenter much easier to be able to just login to an existing EC and execute their OEDL experiment description there, it was decided in this example that the testbed will deploy its own OMF6 EC. Note that OMF6 also requires the presence of an XMPP server for messaging, which was also added to the testbed infrastructure.

As depicted on the slide, the experiment control is then executed as follows:

1. The experimenter gains access to the server that runs the OMF6 EC
2. The experimenter starts the experiment control procedure by giving the command “OMF exec”. The name of the scenario description that is to be executed is given as an argument. The EC will process this description, and will initiate specific commands at certain resources at the appropriate time.
3. To trigger the RC, an XMPP message is sent from the OMF6 EC to the OMF6 RC. This message is compliant with the FRCP protocol

Needed steps to support F4F monitoring

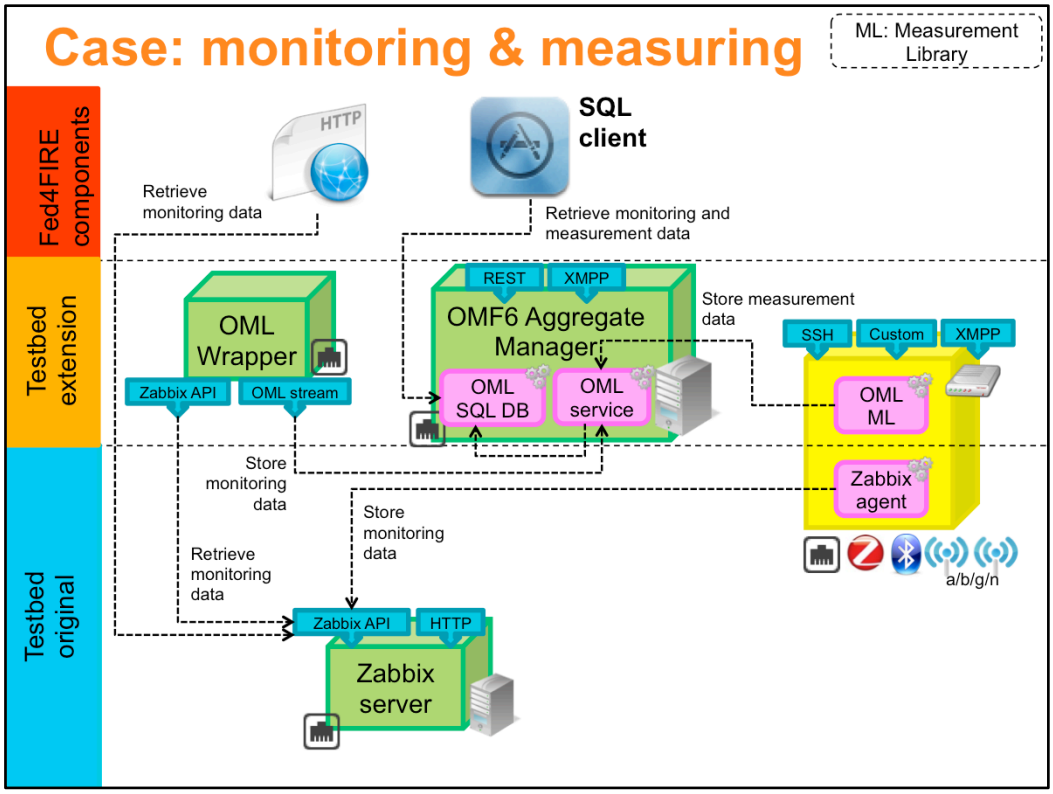
Functional element	Implementation strategy
Facility and Infrastructure monitoring	• Deploy Nagios and/or Zabbix and/or collectd if not yet available • Instrument these relevant measurement systems (all participants, with support from Fed4FIRE)
Experiment measurement	• Deploy OML if not yet available • Instrument relevant measurement systems (all participants, with support from Fed4FIRE)

FED4FIRE

43

iMinds CONNECT.INNOVATE.CREATE

In order to implement the Fed4FIRE design for monitoring as previously explained, seven implementation steps have been identified, as presented on this slide. Most steps (service deployment and application instrumentation) need to be undertaken independently by all participants. Where commonalities exist (e.g. Zabbix and Nagios) instrumentation is a common effort. To support with the instrumentation task, a clearinghouse of homogenized OML measurement points schemas is provided by Fed4FIRE. The goal is to ease integration of new applications while maintaining the highest level of data exchangeability and interoperability between measurement tools providing the similar information.



As mentioned before, we have defined three types of monitoring in Fed4FIRE: facility monitoring (is the facility up and running?), infrastructure monitoring (info about resources, e.g. CPU load, free RAM, etc) and experiment measurement (end to end delay, etc). And in this example testbed Zabbix was already deployed as the monitoring framework responsible for both facility and infrastructure monitoring. To transform this data into the Fed4FIRE common format, OML streams, an additional service needs to be deployed that queries this Zabbix server and translates the stored data into an OML stream. The implementation of this service is done by the specific testbed operator team, but they can use existing scripts provided by Fed4FIRE that illustrate how a Zabbix server can be queried through the Zabbix API and how this fetched data can be injected into an OML stream. This stream is then forwarded using XMPP (not depicted in order not to overload the slide, XMPP server will be available since it was part of the testbed extension for experiment control) to the OMF Aggregate Manager that runs the services needed to persist the data in an SQL DB. This DB is run as part of an OMF6 Aggregate Manager. This is the default way for the installation of an OML persistence service as provided in the default OML package.

For experiment measurements, OML is needed by Fed4FIRE, therefore it is also needed to deploy an appropriate OML Measurement Library ML) on each resource. Since our resources are all Intel x86 compatible nodes running Linux, there was no need to implement a new OML ML, the appropriate one is already existing as part of the standard OML framework. On every resource the Measurement Library can be used by the experimenters to automatically store their measured data in the OML SQL DB of the AM.

Both the monitoring and the measurement data can be retrieved from the OML SQL database using any SQL client. This does not interfere with the original modus of operandi of the testbed: it remains possible to retrieve the monitoring data by surfing to the Zabbix web interface.

Summary

- Fed4FIRE allows facility providers to
  - Reach a larger community of potential experimenters
  - Increase the technical possibilities for their experimenters
  - Lower their costs
- It is based on a distributed architecture, in which common API's are very important
- Several options exist for the adoption of each of them. Default solutions are provided in order to minimize the needed effort to join the federation.

Questions

info@fed4fire.eu

Acknowledgement

- This work was carried out with the support of the Fed4FIRE-project (“Federation for FIRE"), an Integrated project funded by the European Commission through the 7th ICT-Framework Programme. (318389). It 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.