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

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

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

# Deliverable D2.2
# General Architecture and Functional Blocks

Version 1.0

| | |
|---|---|
| **Due date**: | 31/12/2013 |
| **Submission date:** | 11/02/2014 |
| **Deliverable leader:** | PSNC |
| **Author list:** | Radek Krzywania (PSNC), Wojbor Bogacki (PSNC), Bartosz Belter (PSNC), Kostas Pentikousis (EICT), Tom Rothe (EICT), Matthew Broadbent (EICT), Gino Carrozzo (NXW), Nicola Ciulli (NXW), Roberto Monno (NXW), Carlos Bermudo (i2CAT), Albert Vico (i2CAT), Carolina Fernandez (i2CAT), Tomohiro Kudoh (AIST), Atsuko Takefusa (AIST), Jin Tanaka (KDDI), Bart Puype (iMinds) |

**Dissemination level**

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

<THIS PAGE IS INTENTIONALLY LEFT BLANK>

# Table of Contents

# List of Figures

## List of Tables

# Abstract

The deliverable defines general architecture and functional blocks which provides a FELIX Federated Framework for integration of different network resources distributed in a multi-domain heterogeneous environment. The document focuses on all components and their functionalities, including aspects of resources management, configuration, monitoring and user access. The document also specifies mechanisms that are used to provide federated services. This deliverable is the basis for the work related to the implementation of the FELIX Inter-Islands Connectivity Framework.

# Executive Summary

Deliverable D2.2 defines the control and management architecture for federated Future Internet testbeds. Although architectural choices concern mainly a federation of European and Japanese testbeds, and specifically OFELIA and RISE, the project ambitions go far beyond, trying to propose a federation framework to be well adopted by other platforms, not only located in Europe or Japan.

The deliverable challenges a number of questions, which build together a set of components used to construct the FELIX architecture:

- *What are the system requirements for a federation platform?* This deliverable reports on further analysis of the User Requirements previously identified and described in D2.1 [20], which have been translated into System Requirements – consolidated and prioritized FELIX framework requirements, which must be taken into account while the defining the FELIX architecture.
- *What are the existing federation frameworks and control/management tools to be adopted in FELIX?* This deliverable provides an analysis of existing solutions, already deployed in Future Internet testbeds in Europe and Japan. It provides a survey of European and Japanese testbed architectures, with particular focus on federation mechanisms to be re-used in FELIX.
- *What are the key system components of the desired federation platform?* The deliverable introduces functional decomposition of the FELIX architecture. All components of the architecture and their expected behaviour are described in detail, highlighting the functionality of interfaces between external entities and the system itself.
- *How to integrate different resource types for creating a slice out of federated resources?* The deliverable indicates several resource types which are crucial for creation of inter-domain multi-technology slices built out of federated resources. It also explains how to manage those resources and unite them into single slice entity.

The deliverable presents all components, interface and protocols at a high level of abstraction. It states a general view on architecture, building blocks, resource types and collaboration of all system entities for slice delivery, however it does not define the explicit way of software development. This concept is introduced intentionally, giving a level of freedom to software developers implementing the FELIX architecture suited to a specific testing environment. The document is intended as a set of architectural guidelines for developers, rather than the detailed software design of the FELIX platform. In order to make the architecture applicable not only to FELIX project, the document focuses on requirements, rather than on explicit protocols or interfaces to be used. Despite the face that it contains some suggestions (e.g. usage of NSI CS standard or OpenFlow protocol for SDN control), the readers of the document may use any other technologies, as long as they are in alignment with the architecture requirements defined. The FELIX project team will make the final selection of protocols and software tools during the implementation phase, and will deploy the service prototype according to the specific needs and environment, i.e. taking into account the FI experimental infrastructures which are part of the consortium.

This document defines the System Requirements, which are functional and non-functional requirements identified by FELIX partners, needed to implement the framework and run the Use Cases defined in D2.1 deliverable [20]. These requirements are used to define the three levels of FELIX framework management structure: Resource Orchestrators (RO), Resource Managers (RM) and resources themselves. The ROs form a hierarchical tree structure which allows information to propagate in an organized manner and locate resources in multiple federated testbeds. The ROs at the very bottom of the tree control RMs, which are responsible for the configuration of resources in a single domain. Each RM is responsible for a different type of resource. The types are not limited to, but include the following types defined within this document: Transit Network, SDN, and Computing Resources. This organized tree structure is called a FELIX Space and is dedicated to managing the infrastructure

and configuring the user slices within it. When a slice is created, a user can control and access it via the User Space, which consist of various tools provided either by a user or the FELIX project itself (e.g. a SDN Controller). Through the analysis of existing solutions, the use and selection of existing out-of-the-box tools and concepts is proposed. This minimizes the implementation work, and facilitates the re-use of already available products and/or standards, while the emphasis in implementation effort will be placed on integration of federated resources.

This document is addressed to network specialists, network architects and decision makers involved in the construction of SDN testbeds in different phases (from architectural design, through implementation to operation), as well as those software developers implementing specific features of control/management frameworks for SDN testbeds.

# 1 Introduction

The FELIX project aims at creating a common framework in which users can request, monitor and manage a slice provisioned over distributed and distant Future Internet experimental facilities. This document specifies the FELIX architecture, main functional blocks of the architecture, as well as providing a common background of terminology to be used in all documents produced in the project.

In FELIX project deliverable D2.1 [20], six use cases have been identified, detailed and further grouped into two major groups: Data Domain use cases and Infrastructure Domain use cases. These use cases have been identified with the primary objective of motivating concepts and innovations expected through FELIX and to identify and address existing issues and barriers when federating distributed, geographically distant testbed facilities.

The **FELIX Data Domain** use cases mostly target the area of SDN and dynamic interconnections via NSI. Data caching, fast delivery, streaming and the related workflow management are key in this group of use-cases:

- Data on Demand – delivery of distributed data by setting data flows over the network
- Pre-processing and delivery of nearly real-time [satellite] data to geographically distant locations (from EU to JP and vice versa)
- High quality media transmission over long-distance networks

The **FELIX Infrastructure Domain** use cases focus more on the efficient use of federated and dispersed FI resources (on different continents), to migrate entire workloads (VMs and data) or virtual infrastructures in a more efficient way (e.g. with energy saving targets) and enhanced features (e.g. data/service survivability in case of disasters):

- Data mobility service by SDN technologies
- Follow-the-sun / follow-the-moon principles
- Disaster recovery by migrating IaaS to a remote data center

The six FELIX use cases have been further translated into a set of user requirements (UR) that describes the expectations from the FELIX system in terms of objectives, use-case environment, constraints and measures of effectiveness and suitability. This deliverable reports on further analysis of the previously identified user requirements and presents the refined list of system requirements (SR), consolidated and prioritized FELIX framework requirements, which must be taken into account while defining the FELIX architecture. System requirements, which are identified and presented in details in this deliverable, build a foundation for the technical analysis of the system/platform to be designed and implemented in a distributed testbed environment spanning among Europe and Japan.

This deliverable attempts to define a preliminary high-level FELIX architecture. The architectural work on specific FELIX components have been preceded with a deep analysis of current architectures and existing testbed management/control frameworks running in Europe and Japan to re-use as much as possible and avoid re-inventing already established and well working mechanisms and algorithms. Project partners agreed to restrict the analysis to the OFELIA, FIBRE, Fed4FIRE and BonFIRE projects on the European side and the GridARS and RISE projects on the Japanese side. The state of the art analysis focused, among the others, on the following architectural components:

- General control frameworks
- Resource discovery, reservation and provisioning mechanisms
- Experiment managers
- Identity management tools

- User interface tools

It should be noted that all the aforementioned projects run their own complete stack to manage resources in either distributed or centralized testbed environments. Each system component of the existing frameworks has been analysed, highlighting its usefulness for FELIX. The resulting FELIX architecture clearly marks the expected progress beyond state of the art, while a decision on which component from the existing frameworks will be re-used during the implementation phase will depend on the final software design, which will happen after completion of this architectural framework.

FELIX aims to provide a framework for integration of different network resources residing in a multi-domain heterogeneous environment. The resulting architecture should be flexible and scalable to assure a sufficient level of interaction between various system components. In the FELIX project it is agreed to use a hierarchical model for inter-domain dependency management, with orchestrator entities responsible for synchronization of resources available in particular administrative domains.

The architecture can be seen as a combination of two spaces (see Figure 1.1):

- **FELIX Space** consists of management and control tools to coordinate processes of creation of a virtual environment in a heterogeneous, multi-domain and geographically distant testbed. The components of the FELIX space will operate in hierarchical model, to enable efficient information management and sharing across multi-domain environment.

- **User Space** consists of any tool or application a user wants to deploy to control his virtual network environment or to execute a particular operation within it. The selection of tools is completely dependent on specific user requirements and is out of scope of the FELIX framework.

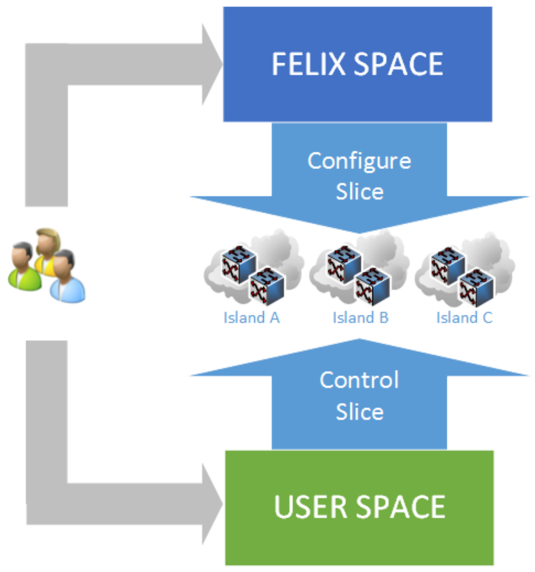

Figure 1.1: FELIX and User Spaces Architectural Concept

Both spaces play distinct roles in creation and operation of each virtual network environment, which is created by the FELIX Space upon a request from a user, and then managed by user tools in the USER Space.

In this architectural document, all components, interfaces and protocols are described at a high level of abstraction. Consequently, the document provides a set of architectural guidelines for software developers, while eventual decisions on implementation choices (e.g. specific technologies) to be used will be made while the detailed software architecture is being realized. The document is structured as follows:

- System Requirements chapter contains an explanation of functional and non-functional requirements that must be fulfilled by the FELIX framework, in order to operate and allow to implement the Use Cases, as defined in [20].
- Related Work and Testbed Analysis chapter contains a brief overview of FI architectures that we selected as related to the FELIX work. The chapter also list tools and technologies which FELIX may potentially utilise and also explain the advantage of new framework over the existing ones. The more detailed FI testbed descriptions can be found in Appendix A.
- System Architecture chapter provides an overview of FELIX architecture details and components. This chapter is divided into the following sub-chapters:
  - Concept and Definitions -- explains terminology and concepts used in this document, and consolidates the preliminary terminology defined in [20].
  - Architecture of the Components -- explains the FELIX and User Space components and their dependencies, interactions, interfaces and responsibilities.
- Conclusions and Summary chapter contains a summary of the FELIX architecture, conclusions and concepts for further development.

# 2 System Requirements

A virtual infrastructure or a slice, distributed over multiple regions and domains, enables large-scale data intensive scientific computing, such as high energy physics, bioscience and geoscience, and highly available and performance assured commercial services. Data intensive scientific applications need to manage Peta-to-Exascale data, produced by geographically distributed high performance experimental instruments and sensors, provided by several organizations, and process them effectively. In addition, commercial service applications require stable distributed computing infrastructure because they have to provide national and worldwide users with a quality-assured service, and prepare for recovery and continuation of the services after a natural or human-induced disaster.

Crucial issues to provide a slice in a federated testbed environment are as follows:

- Resource Orchestration -- Orchestration of various virtualised resources, not only computers, but also network and storage, provided from multiple domains, is required.
- Island Resource Management -- Coordination of various resources provided by heterogeneous resource management systems within an Island is required.
- Resource Allocation Planning -- It is important to create a suitable resource allocation plan for both computing resources and network resources. This should take into consideration user and resource administrator issues, such as cost, energy consumption and load balancing.
- Provisioning -- It is important to provide applications with a virtual flat environment, just like a dedicated cluster, using dynamic resource information, such as IP addresses.
- AAA (authentication, authorization and accounting) -- It is vital, that all actions are proven to be performed by authorized actors -- and only by those authorized. This includes making sure the persons are who they claim to be, ensuring people may perform the actions they are trying to and to record those actions.
- Monitoring -- It is difficult for each user to monitor the usage of distributed and heterogeneous "virtual" resources managed by multiple domains. Each domain has to provide monitoring information for resources (which are virtual rather than physical) which are parts of a slice belonging to the user. Such monitoring information from multiple domains has to be coordinated and provided to the user.

Recently, cloud computing or IaaS (infrastructure as a service), in which a “slice” is constructed dynamically according to a request, is coming into widespread use.

A slice is an infrastructure constructed on top of physical resources (such as computer and storage hardware) using virtualisation technologies. A user to whom a slice is provisioned can use it as if it was his own physical infrastructure. Using such dynamic slice, from the view point of a user, a required infrastructure (such as computers and storages) is provisioned dynamically when it is needed. Users do not have to own their own resources, and should pay the cost only when they use the resources. From the view point of resource providers, utilization of the resources can be maximized, and the operating cost can be minimized by having a large number of uniform resources at one place. In addition, the energy consumption of a slice can be reduced by optimizing resource usage. In existing Cloud Management Systems (e.g. OpenStack, CloudStack), all the physical resources, which are composed in one slice, belong to a single data center. Therefore, optimization (in terms of both performance and energy cost) of resource utilization among multiple data centers is not possible.

Different services such as computing and storage may be supplied by different providers. In such a case, network bandwidth between data centers of the providers is important to achieve high performance and stable service. By providing a wideband stable network between data centers, an inter-domain slice can be provisioned, therefore being composed of resources delivered by multiple data centers.

In an inter-domain slice, an inter-cloud network provided dynamically is connected to an intra-data center network. Since there will be multiple tenants of cloud services in a data center, multiple virtual networks will be formed inside of an intra-data center network. It is important that these virtual networks are properly connected to inter-data center networks; consequently, inter-data center resource management becomes important, indeed a number of research results have been reported on this issue. A Discovery Service, which collects static resource information, a Resource Management Service, which schedules and co-allocates appropriate resources, a Provisioning Service, which constructs a virtual infrastructure for the resources at a reserved time and a Monitoring Service, which collects resource usage information of each user's slice and provides it to the user are required.

For all actions, the FELIX architecture and implementation must make sure that all actions are performed by authorized actors only. All entities with their respective interfaces need to adopt a authentication and authorization scheme. This scheme needs to ensure that the facility's entities, including their interconnections, are protected from malicious attackers. Especially, the management interfaces need to be safeguarded with authentication and authorization measures. Also, accounting is a vital part of security, because the logging of actions can later not only be used to charge the perpetrator, but also to learn and guard for the future.

This chapter will present the FELIX framework system requirements (SR), which contributes to the proposed architecture approach and fulfills the User Requirements defined in D2.1 Experiment Use Cases and Requirements deliverable [20]. The Software Requirements (SRs), in comparison to Use Cases and URs (User Requirements), identify particular features of the architecture and software modules, creating a list of requirements facilitating the validation of the final FELIX product. SRs are more detailed, often splitting single URs into several functional requirements, realized by one or more dependent software modules. The requirements presented in this section are grouped in three sets, depending on the assessment made by the FELIX partners on the importance of the particular requirement for the implementation of a use case. This importance ranking is expressed according to [14] with the following keywords:

- MUST – for requirements which are mandatory for design and implementation of the FELIX framework and its components,
- SHOULD – for requirements which are recommended due to increased efficiency, optimization or any other positive effect to the end users
- MAY – for requirements which are optional and do not influence overall FELIX functionality in significant manner.

SRs usually directly reference one or more URs, explicitly linking the Use Cases with specific FELIX functionality.

| ID | Requirement | Description | UR Ref. |
|---|---|---|---|
| SR.1.1 | User request acceptance | The FELIX framework MUST accept user requests for a slice, which includes a minimal set of details required to create a multi-technology multi-domain slice. The "minimal set of details" is technology dependent and is not in scope of this document. The approval of a single request may be the subject of additional considerations according to set policy or system constraints. | |

| | | | |
|---|---|---|---|
| SR.1.2 | Implement a slice | The FELIX framework MUST implement a user slice, according to resources availability and various constraints, respecting slice requirements provided by a user. The slice may be multi-technology and multi-domain. FELIX MUST be able to orchestrate the resources allocation process and reply user with success or failure notification of slice creation. | |
| SR.1.3 | Request API | The FELIX framework MUST provide a user API, which allows to perform the minimal set of operations: (i) request a slice; (ii) check slice status; (iii) request a slice termination | |
| SR.1.4 | Distributed environment support | The FELIX framework by default is a multi-domain environment and therefore its components MUST be designed and implemented in a way allowing the distribution of the components and independent system deployments in particular domains. All framework entities MUST be able to be instantiated as standalone components, and MUST NOT depend on each other. The only allowed dependency is a network communication between entities, however entities should handle communication failures and continue operation without critical errors. | |
| SR.1.5 | User slice control | A user MUST have immediate access to a created FELIX slice, so that he/she can reach and manipulate any resource, that was expressed as a slice requirement. FELIX may hide some slice infrastructure components, which are vital for slice delivery but was not explicitly mentioned by a user, e.g. network hardware for inter-domain connectivity, hardware virtualization platform, etc. | |
| SR.1.6 | User web portal | The FELIX framework MUST provide end users with a graphical user interface, e.g. a web portal, as an interaction mechanism. A user must be able to manage his slices and reservations through the GUI and receive notification from the system. | |
| SR.1.7 | Command line access | The FELIX framework MUST provide users with command line style interface, in order to request and manage slices. It is advised to reuse existing CLI tools, like OMNI or SFI, with proper adaptation to FELIX architecture. | |

| | | | |
|---|---|---|---|
| SR.1.8 | Complex slice infrastructure creation | The FELIX framework MUST be able to build on demand a complex slices, using infrastructures and resources of different domains at the same time. Users should not be restricted by the architecture in the amount of resources allowed to build a slice, except for authorization, policy and infrastructure constraints. Slices may consist of any amount of SDN, transport network and/or IT resources, providing users with wide range of services and giving the control over the resources directly to a particular user. The exemplary usage of FELIX framework is described in [20] deliverable in Use Case section. | |
| SR.1.9 | Messaging consistency and integrity | The message exchange between the FELIX framework entities MUST be assured to be secure and consistent, in the sense that message delivery must be controlled and monitored. The entities must be assured that the message is delivered or a failure has occurred. The integrity of the message must be protected, in order to prevent modification of the message content by unauthorized external entities. | |
| SR.1.10 | The FELIX framework must control resources of different types | In order to deliver to a user a multi-technological slice, including resources of different types, the FELIX framework MUST be able to manage different kind of technologies, including SDN, transport networks, and IT resources. FELIX MUST provide mechanisms to request configuration and synchronize technological parts of slice. | UR.1, UR.2 |
| SR.1.11 | Scalable technology modules deployment | The technology management modules (Resource Managers -- RM) in single domain MUST be able to be deployed in scalable and efficient way. The particular technological sub-domains managers should be able to be deployed independently and should not relay on each other during operation, unless a synchronization effort is needed. Therefore an administrator should have an option to deploy only some of the available RMs, and not all of them, if they are not required. | |
| SR.1.12 | Support for SDN resources | The FELIX framework MUST be able to configure SDN resource types within a particular domain, in order to create and configure a user slice, and deliver this slice under user control. | UR.3 |
| SR.1.13 | Support for Transport Network resources | The FELIX framework MUST be able to configure a transport network resource types within a particular domain, in order to create and configure a user slice, and deliver this slice under user control. | UR.3 |

| | | | |
|---|---|---|---|
| SR.1.14 | Support for IT resources | The FELIX framework MUST be able to configure an IT (e.g. servers, data storage, etc.) resource types within a particular domain, in order to create and configure a user slice, and deliver this slice under user control. | UR.3 |
| SR.1.15 | Orchestration and synchronization of resources configuration | The FELIX framework MUST orchestrate and synchronize configuration of particular sub-domain configurations (SDN, transport networks, IT, etc.), providing unified slice resources able to collaborate in a transparent way (not disturbing end user actions). Particularly, the integration of SDN and Transport Network resources is critical for achieving the FELIX project objectives. | UR.3, UR.10 |
| SR.1.16 | Organized orchestration layer of FELIX framework | The orchestration layer of FELIX MUST be hierarchical, organized, and scalable in the context of deployment and management. This will require coexistence of multiple orchestration entities, able to collaborate in organized manner for:<br>• delegating (also splitting) requests to appropriate orchestrators or RMs<br>• forwarding messages in hierarchical model<br>• synchronize state for consistent global view of slices and resources | UR.11 |
| SR.1.17 | Resources allocation mechanism in distributed environment | The FELIX framework MUST have a resources allocation mechanism, which will be able upon a use request to:<br>• identify required resources<br>• locate the resources in domains (relaying on available information)<br>• construct initial draft of slice description, including information on domains and their resources required to build a user slice<br>• accept constrained queries providing slice descriptions excluding particular resources in a domain (e.g. when a reservation fail due to particular domain, a new resources search should not try to use the ""refused"" resources ) | UR.4 |

| | | | |
|---|---|---|---|
| SR.1.18 | Support for inter-domain transit network configuration | Transit Network resources MUST be able to be configured involving multiple domains, so that a created connection can link and/or pass multiple administratively independent domains, managed by different ROs and RMs. The inter-domain configuration must be under control of proper ROs and RMs, which should be aware of such action and required interaction. | UR.6, UR.12 |
| SR.1.19 | Authentication of users | The FELIX framework MUST be able to authenticate end users in advance, before providing them access to any FELIX controlled resources and slices. | UR.5 |
| SR.1.20 | Authentication of FELIX entities | All FELIX entities MUST be able to authenticate each other (e.g. with X.509 certificates), in order to prevent unauthorized resource manipulation. | UR.5 |
| SR.1.21 | Authorization of users | The FELIX framework MUST be able to authorize end users on particular resources usage in order to prevent abuse of resource usage. Users should have access only to the resources assigned to them by FELIX framework. Authorization data will also prevent users from invoking unauthorized actions and define users roles (e.g. administrator, experimenter, etc.) | UR.5 |
| SR.1.22 | Authorization of FELIX entities | All FELIX entities MUST be able to authorize each other , in order to prevent unauthorized resource manipulation. | UR.5 |
| SR.1.23 | Users notifications | Users MUST be notified about significant events happening in the FELIX framework environment. Such events include:<br>• acceptance of a single slice request<br>• confirmation of resource booking<br>• notification on slice set up<br>• notification on recognized failures<br>• confirmation of slice tear down<br>• notification of administrative messages (e.g. planned maintenance)<br>The notification may be delivered at least as:<br>• email sent to a user, if a user explicitly express such interest while request submission and provide system with a valid email address<br>• a notice in GUI related to a user, a slice or a particular reservation. | UR.8 |

| | | | |
|---|---|---|---|
| SR.1.24 | Resources awareness | The FELIX framework MUST be aware of all resources available for configuration in the whole controlled environment. This information is vital for proper operation, resource management and configuration. This information must be organized and structured allowing browsing resources types, domains in charge of them, and relation between domains (e.g. network connectivity). | UR.7 |
| SR.1.25 | Resource information propagation | The global resource information MUST be dynamically distributed to all FELIX entities, which consider this information as critical to operate (mostly ROs and resources allocation entities). The framework MUST deliver an automated mechanism (e.g. a lookup service) where resource information can be stored and distributed in consistent state. | UR.7 |
| SR.1.26 | Resources usage tracking | The FELIX framework MUST keep track of resources usage, assignment, and availability in order to search and allocate only free resources to new slices. FELIX MUST guarantee that the same resource is not shared between more than one user at the same time, providing exclusive resources access and isolation of slices. | UR.7 |
| SR.1.27 | End users VPN service | The FELIX framework MUST provide a VPN service with configurable resource access, limited to slice scope. This will be one of the mechanisms providing isolation of user slice and default way for users to access their slice resources. The configuration of VPN service per user must include setting up authentication and authorization details, restricting access to allowed resources only, setting up access policies, and possibly firewall restrictions. The user must be authenticated and authorized to the VPN service in order to prevent resources abuse or unauthorized access. The VPN service is configured on per slice and per user basis, and the configuration process is allowed to be manual. | UR.9 |

Table 2.1: MUST System Requirements

| ID | Requirement | Description | UR Ref. |
|---|---|---|---|
| SR.2.1 | Accounting information | The FELIX framework SHOULD be able to collect accounting information in order to track user activity and resources utilization. This will allow create resources usage reports and account users for their resources utilization. Accounting information may also be used to restrict users e.g. with resources quotas, etc. | UR.20 |

| | | | |
|---|---|---|---|
| SR.2.2 | Monitoring up-to date resources status | The resources information and/or FELIX system entities SHOULD be able to be updated frequently or on-event by monitoring system, in case of resource status change, i.e. in unexpected failure conditions. This will potentially allow to make FELIX framework more robust and react on critical situations. | UR.13 |
| SR.2.3 | Notify users on unexpected failures | The FELIX framework SHOULD be able to notify users about critical situation encountered during slice operation, i.e. unexpected resources failure. The system may or may not undertake a repair action, however user SHOULD be notified that the slice is not fully operated or unavailable at all. The notification SHOULD include the amount of information will not discover FELIX critical information, yet it will be meaningful for the end users, allowing to understand the cause and location of the problem. | UR.8, UR.13 |
| SR.2.4 | Resilient service configuration | While searching, allocating and configuring FELIX resources for slice purposes, FELIX framework SHOULD be able to implement resiliency features, which will protect all or critical resources in case of failure. Resiliency SHOULD be optional for end users and must be explicitly expressed by an end user at slice request time. | UR.17 |
| SR.2.5 | Critical failure restoration | In case of a critical failure regarding usage of slice resources, the FELIX framework SHOULD take a repair action. If a user expressed resiliency expectation, the protected/backup resources can be used. If a user did not express resiliency as a requirement, or failure applied to non protected resources, a repair action MAY involve reconfiguration of existing slice pre-empted by now resources search with additional constraints. | UR.17 |
| SR.2.6 | Optimization and automation of resources allocation | The resources information and/or FELIX system entities SHOULD be able to be updated frequently or on-event by monitoring system, in case of resource status change, i.e. in unexpected failure conditions. This will potentially allow to make FELIX framework more robust and resilient, and as a consequence provide better, more reliable services to the end users. The information on current and planned resources status can be potentially also used for resource scheduling and planning, in order to make the assignment more stable and failure proof. | UR.15 |

| | | | |
|---|---|---|---|
| SR.2.7 | Dynamic resources configuration | The FELIX framework resources allocation mechanism SHOULD be automated, so that the incoming request are served immediately without delays of human intervention. Framework should have autonomy to decide which resources and in which domain should be delegated in order to build a user slice. The process should include optimization mechanisms and respect user constraints and SLA. | UR.15 |
| SR.2.8 | Automated configuration of VPN service | The FELIX framework VPN service SHOULD be configured automatically by FELIX framework on per slice per user basis, regarding a user slice request and assigned resources. The framework should have all credentials to provide authentication and authorization configurations, set up firewall policies, user policies, access privileges, and whatever other actions, required to allow particular user to access his assigned resources/slice. At the same time the configuration must assure slice and users isolation in the FELIX environment, preventing resources abuse and over-utilization. | |
| SR.2.9 | RMs may be able to interact directly in inter-domain environment | For optimization and efficiency of configuration process RMs SHOULD be permitted to interact, despite they may be deployed in different administrative domains and be supervised by different ROs. In case a configuration of one resources depends on configuration of other resources in different domain, and those resources are of the same type, the RMs can interact directly, without intermediate supervising ROs. This situation can occur e.g. while setting up transport network connectivity and domains must agree common VLAN identified or exchange other connection specific attributes. | UR.12 |
| SR.2.10 | Resources allocation optimization | While searching and allocating resources, the FELIX framework SHOULD use mechanisms allowing optimization of resources utilization, which are not explicitly mentioned in a user request. Such mechanism may e.g. consider load balancing, resources utilization levels, overall energy consumption, etc. | UR.14 |

| | | | |
|---|---|---|---|
| SR.2.11 | Request attributes | A user slice request must contain a minimal required set of attributes as defined by FELIX. The FELIX framework SHOULD however allow to specify also optional attributes, which constraint a slice resources in more advance manner. For transport network connection a minimal set of information to deliver a circuit is a start point, end point and capacity, while user can additionally request e.g. RTT limits, or allowed VLAN range. For SDN resource types, a user may want to define details regarding traffic organization specifying particular network flows or restrictions on SDN resources. | UR.15, UR.16, UR.19 |
| SR.2.12 | Default slice controller | The FELIX framework SHOULD deliver a default slice controller, which can be used by a user to manipulate resources within a particular slice. A user however is not obliged to use this controller and may deploy its own. The decision must be however taken at the slice request submission time, as proper resources holder will be prepared. | |
| SR.2.13 | Advance reservations | The FELIX framework SHOULD support advance reservation scheduling, where users can specify a slice start time in the future and a lifetime duration of a slice. This will force the framework to analyse resources availability not only in the moment of serving the request but also future planning of resources usage. The time constraints for user (e.g. the duration of a reservation and how far in advance can a reservation be requested) should be defined in the form of service policy and framework should take them as configuration item. | UR.18 |

Table 2.2: SHOULD System Requirements

| **ID** | **Requirement** | **Description** | **UR Ref.** |
|---|---|---|---|
| SR.3.1 | Multi-point to multi-point network connectivity | The FELIX framework MAY support multi-point network connectivity for slice building. The default method for implementing a transport network connectivity, i.e. for multi-domain purposes, is a point-to-point service. Providing multi-point services will enable more scalable configuration and allow more advanced slice configuration. | UR.21 |
| SR.3.2 | Data replication mechanism | Usage of multiple data storage facilities in single slice may potentially require a synchronization of data repositories. The FELIX framework MAY support such synchronization and provide data replication tools internally. | UR.23 |

| | | | |
|---|---|---|---|
| SR.3.3 | Monitoring API | The FELIX framework MAY implement API of various monitoring tools (e.g. PerfSONAR, Nagios, etc.), which will allow to reuse existing monitoring solutions and improve FELIX monitoring capabilities. | UR.22 |

Table 2.3: MAY System Requirements

# 3 Related Work and Testbed Analysis

This chapter discusses some existing FI initiatives and infrastructures relevant to FELIX and analyses the related outcomes and architectural approach with respect to the FELIX design goals.

## 3.1 Survey of European and Japanese testbed architectures

In this chapter, the main aspects of some FIRE (EU) and RISE (Japan) projects are summarized with focus on the proposed architecture, the interconnections between the different testbeds and the federation approaches. We have chosen to restrict the analysis to the OFELIA, FIBRE, FED4FIRE, BonFIRE projects for the European side and to the GridARS and RISE projects for the Japanese side.

### 3.1.1 OFELIA

OFELIA is a collaborative project within the European Commission's FP7 ICT WorkProgramme. It started in October 2010 and ended in October 2013.

The OFELIA project created, and now maintains, a pan-European experimental network facility that allows researchers to not only experiment "on" a real network but to control and extend the network itself in a precise and dynamic fashion. The OFELIA facility is based on OpenFlow, an emerging networking technology that allows virtualization and control of the network environment through secure and standardized interfaces.

Ten interconnected islands, based on OpenFlow hardware infrastructure, form a diverse OpenFlow infrastructure that allows experimentation on multi-layer and multi-technology networks. OFELIA's objective is to provide experimental facilities which allow for the flexible integration of test and production traffic by isolating the traffic domains inside the OpenFlow enabled network equipment. This creates realistic test scenarios and facilitates the seamless deployment of successfully tested technology into the real-world.

The overall, high level system architecture for OFELIA can be divided into 2 layers -- as illustrated in Figure 3.1. From bottom to top:

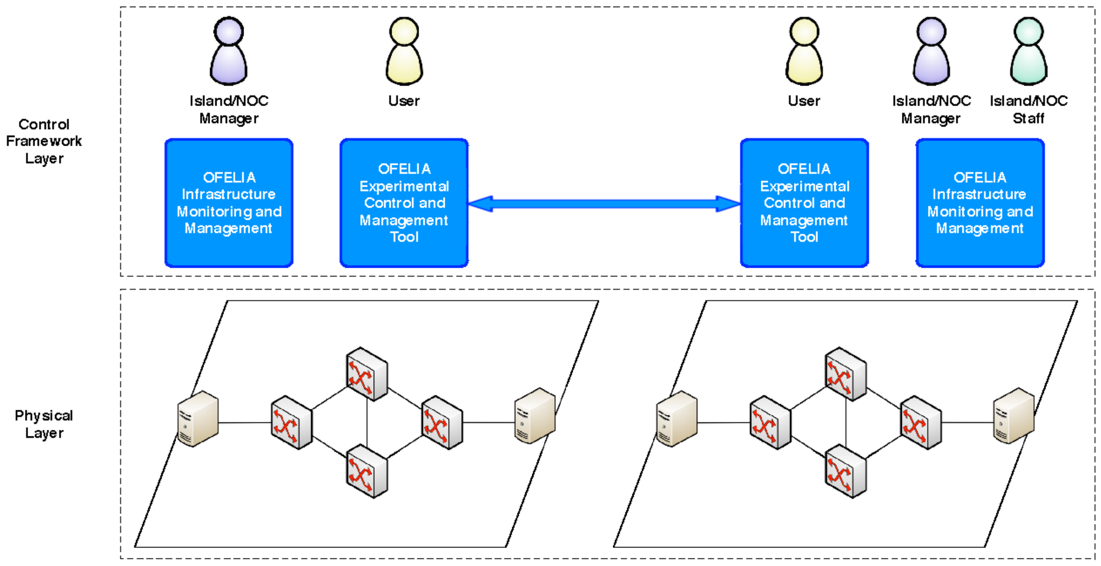
Control Framework Layer
Island/NOC Manager
User
User
Island/NOC Manager
Island/NOC Staff
OFELIA Infrastructure Monitoring and Management
OFELIA Experimental Control and Management Tool
OFELIA Experimental Control and Management Tool
OFELIA Infrastructure Monitoring and Management
Physical Layer

Figure 3.1: OFELIA High Level System Architecture

- **Physical layer**: comprises the computing resources (servers, processors) and network resources (routers, switches, links, wireless devices and optical components). We identify this as the physical substrate, which is composed of these computing and network resources, the connectivity between them and the physical topology itself. These resources are accessed by the OFELIA Control Framework to achieve its experimental facility objectives. Since the technologies and components used in the physical substrate do evolve with time as new innovations arise, OFELIA aims to keep track of the technology evolution and accommodate the appropriate changes to the facility throughout the life time of the facility.

- **Control Framework layer**: contains the whole control framework components, which manage and monitor the applications and devices in the physical substrate layer. The Aggregate Managers and Resource Managers are placed here. This layer can be further divided in its components:
  - the **Expedient** is the GUI and allows the connection and federation with different Aggregate Managers via its plug-ins
  - the **Aggregate Managers** enable experimenters to create both compute and network resources via the VT AM and OF AM respectively
  - the **Resource Managers** do directly interact with the physical layer, provisioning computes (XenServer) or flow rules to establish the topology (FlowVisor)

We could also consider a hidden and higher level in the user flow that is performed by humans. It is there where the policies for the usage of the facility are defined. For that matter, the Network Operations Centre acts as the first point of contact for all technical and non-technical issues relating to the facilities and experimenters/researchers who will use the OFELIA facility, deciding policies on the resources and the grant or denial of requests.

For further details on the architecture, please refer to Appendix A.

### 3.1.2 FIBRE

The FIBRE (Future Internet testbeds and experimentation between BRazil and Europe) project aims to design, implement and validate a shared Future Internet research facility, supporting the joint experimentation of European and Brazilian researchers.

The FIBRE infrastructure is a federation of testbeds distributed across Europe and Brazil. The FIBRE-EU system connects the OpenFlow-based testbeds developed in Barcelona (i2CAT) and Bristol (University of Bristol) which are managed by the OFELIA control framework. Moreover, it incorporates the NITOS testbed deployed at University of Thessaly, which is composed by several wireless nodes based on commercial WiFi cards and Linux open source drivers. On the other hand, the FIBRE-BR testbed includes nine Brazilian partners interconnected using private L2 channels. The VLAN-based L2 physical link between Europe and Brazil is provided by GÉANT, Internet2 and RedCLARA.

The whole infrastructure is managed by different kinds of control and monitoring framework (CMFs). Indeed, FIBRE includes and enhances testbeds from other projects like OFELIA, OMF and ProtoGENI, which has been modified with the necessary software components to align their northbound interface to Slice-Based Federation Architecture (SFA) specifications [4].

The FIBRE architecture is composed by several multi-layer building blocks, as follows:

- **Authorities.** The FIBRE project has chosen to have two top-domain authorities, the first under the responsibility of Brazil and the latter under Europe responsibility. These inter-connected authorities can inter-operate to allow the federation of BR and EU sites.
- **SFA Registry.** The SFA Registry is a database able to store the information related to users and projects. It should manage the certificates provided by the authorities.
- **Portal.** MySLICE tool is the graphical (web) user interface chosen by FIBRE. Refer to Table 3.1 for details.

- **SFA gateway.** The SFA gateway is designed to translate the user's requests to SFA-based requests which are aligned to the GENI (version 2 or 3) API. The SFA gateway also provides slice management functions.
- **Aggregate managers.** FIBRE reuses the Aggregate Managers (AM) already developed in OFELIA projects related to OpenFlow (OPTIN AM) and Xen-based (VT AM) resources and introduces a new AM to manage optical switches (ROADM) devices.

The following Figure 3.2 depicts the FIBRE architecture:

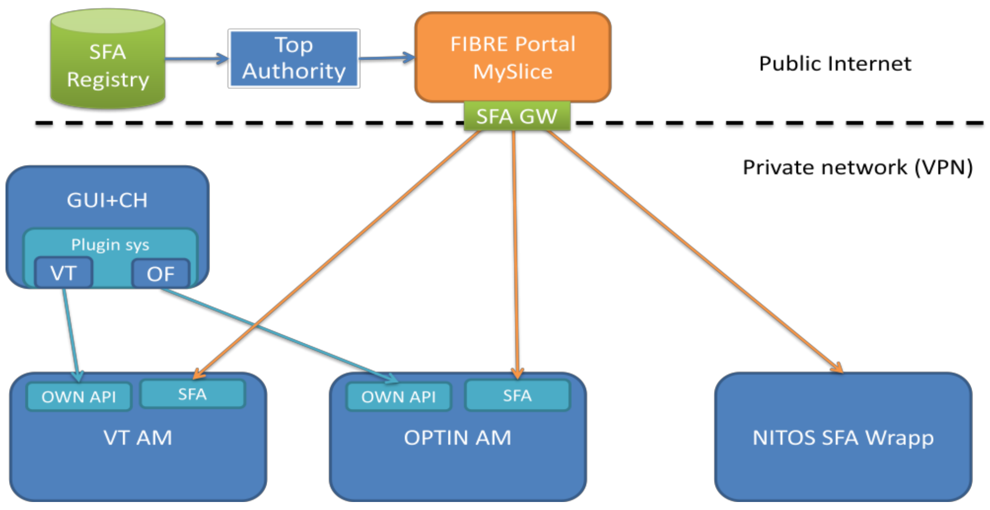

Figure 3.2: FIBRE Architecture

For further details, please refer to Appendix A.

### 3.1.3 Fed4FIRE

Fed4FIRE offers a common federation framework for Future Internet Research and Experimentation facilities that:

- Is adopted by different communities (experimentation facilities, experimenters, academia, industry)
- Supports powerful experiment lifecycle management (including tools for discovery and reservation, experiment control, measurements, etc.)
- Supports key aspects of trustworthiness (federated identity management and access control, accountability, SLA management)

For experimenters, Fed4FIRE facilitates the creation of experiments that break the boundaries of the different FIRE domains (wireless, wired, OpenFlow, cloud computing, smart cities, services, etc.) and easily access all of the required resources with a single unified account. It allows experimenters to focus on your core task of experimentation, instead of on practical aspects such as learning to work with different tools for each testbed, requesting accounts on each testbed separately, etc.

The Fed4FIRE architecture is made up of a number of components. In Figure 3.3, we describe the components of the architecture in a resource discovery, requirement, reservation and provisioning scenario. The architecture consists of 4 layers: testbed resources, testbed management software, broker services and experimenter tools:

- The bottom layer of Fed4FIRE is composed of the the physical testbed resources (servers, virtual machines, switches, sensors, services, etc).

- On top of this, the testbed management software manages these resources. In addition, the testbed users and experiments are present at this level.

- The third tier of Fed4FIRE is a 'broker services' layer which contains services run by 3rd parties or the federation itself. These broker between the testbeds and the experimenters. For example, a broker reservation service will seek to match the resources demanded by an experimenter and the availability of these within the testbed. An orchestration service or a portal is also considered to be a broker.

- At the top of the Fed4FIRE architecture, the experimenter tools/user interfaces are found. These are deployed on the experimenter's computer and are used by the experimenter to communicate with the testbed management frameworks, testbed resources and brokers.

Each software component depicted in the Figure 3.3 has an interface describing how other components can communicate with it. Identical interfaces are annotated with the same color. Therefore when there are different colors present on some components, this means that they expose different interfaces. The arrows between components show the interactions. In the vertical columns of the picture, three administrative domains are envisioned: testbed A, testbed B and the federation facilitator. These three domains refer to logical locations, not physical ones. So testbed A resources can be distributed over multiple locations (e.g. PlanetLab), but the management of that testbed is under a single administration. The same holds for the federation facilitator: components can be distributed over multiple datacenters, but they are under a single administration entity. This however does not exclude the possibility that 3rd parties will arise with additional facilitation functionalities. Besides, it is also possible that the federation facilitator is mirrored across different domains to introduce redundancy, which could be valuable in case of failure or discontinuation of the main federation facilitator.

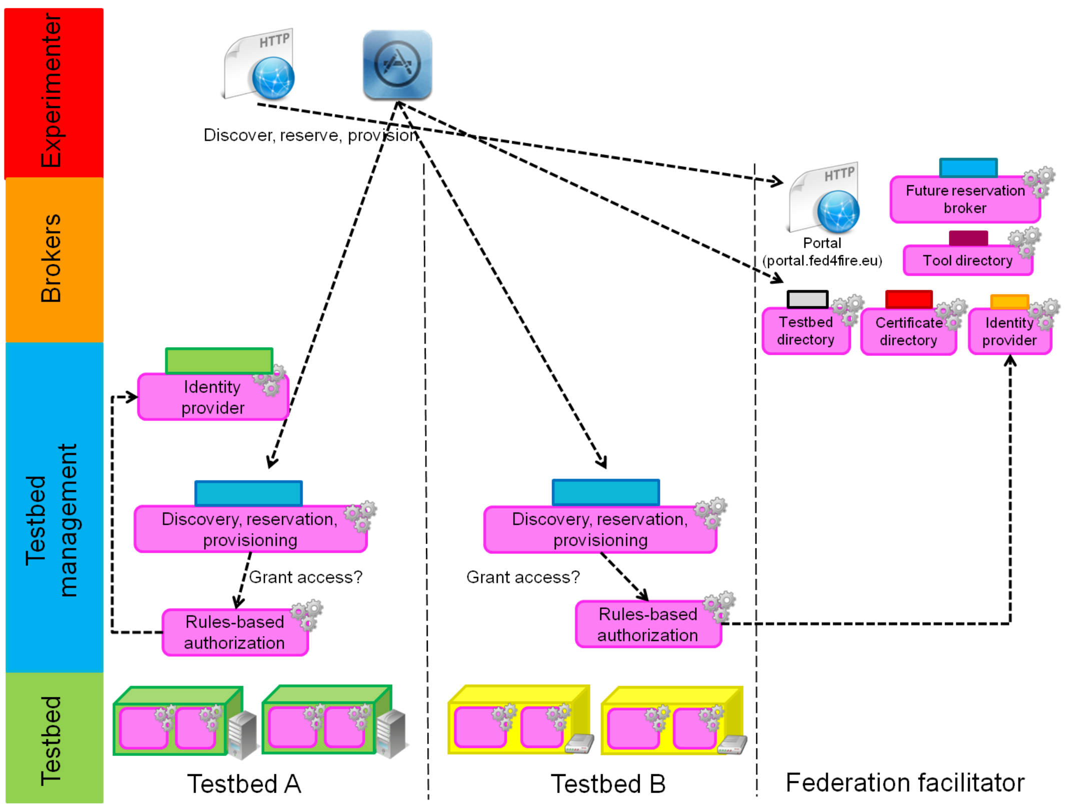

Figure 3.3: Proposed architecture for discovery, requirements, reservation and provisioning

Figure 3.3 also demonstrates how Fed4FIRE has adopted a distributed architectural design. Components can be identified at the testbed location, at the federation facilitator, and at the experimenter level (in this case different experimenter clients tools). One of the main design principles of the architecture is that components belonging to the federation facilitator are only intended to make operation and usage of the federation more convenient. They may never be actually necessary for a correct operation of the federation. F4F call these 'brokers', as they provide 'brokered' access between experimenter tools and the testbeds.

For further details, including information on Fed4FIRE monitoring, measurement and experiment control, please refer to Appendix A.

### 3.1.4 BonFIRE

The BonFIRE (Building service testbeds for Future Internet Research and Experimentation) project provides a state-of-the-art multi-site cloud facility for applications, services and systems research in the Internet of Services (IoS) community. The infrastructure, composed of 7 geographically distributed testbeds across Europe, gives a controlled access to heterogeneous resources (compute, storage and networking) to researchers who can benefit of the necessary control and monitoring tools for a detailed experimentation of their systems and applications.

The BonFIRE framework manages the physical and virtual resources which can be easily created, updated and deleted. Moreover, the available physical hardware (162 nodes/1800 cores) can be automatically configured or reserved "on-demand". The entire system is monitored providing single or aggregated metrics at resource and infrastructure level (e.g. CPU usage, packet delay).

To fulfill these requirements, BonFIRE adopts an architecture composed by several elements which expose their functionalities through well-defined APIs. The main components are the following:

- The **Portal** and the **CLI tools** offer the user interface to request virtual infrastructures and show the running experiments, the available resources at each testbed site, the monitoring information, etc...
- The **Experiment Manager** provides an interface to schedule, plan and orchestrate the execution of an experiment.
- The **Resource Manager** provides an interface to create, manage and terminate compute, storage and network resources, which may physically reside at any testbed in the BonFIRE system.
- The **Enactor** allows the decoupling of the specific implementations of the testbed APIs from the BonFIRE Resource Manager providing a unified interface.

BonFIRE has also addressed the issues related to the interconnections between different sites. Instead of relying on the best-effort Internet connectivity, the cloud resources belonging to different testbeds can be interconnected through a dedicated network system which offers a Bandwidth on Demand (BoD) service. This service is provided through the interconnection between some BonFIRE testbed sites and the GÉANT Bandwidth-on-Demand system (AutoBAHN). BoD services are described in BonFIRE through a new OCCI resource (the site-link) and they are managed through a dedicated adaptor implemented within the enactor component.

The Figure 3.4 gives an overview of the adopted BonFIRE architecture which includes the Cloud-to-Net (AutoBAHN) services. For further details, please refer to Appendix A.

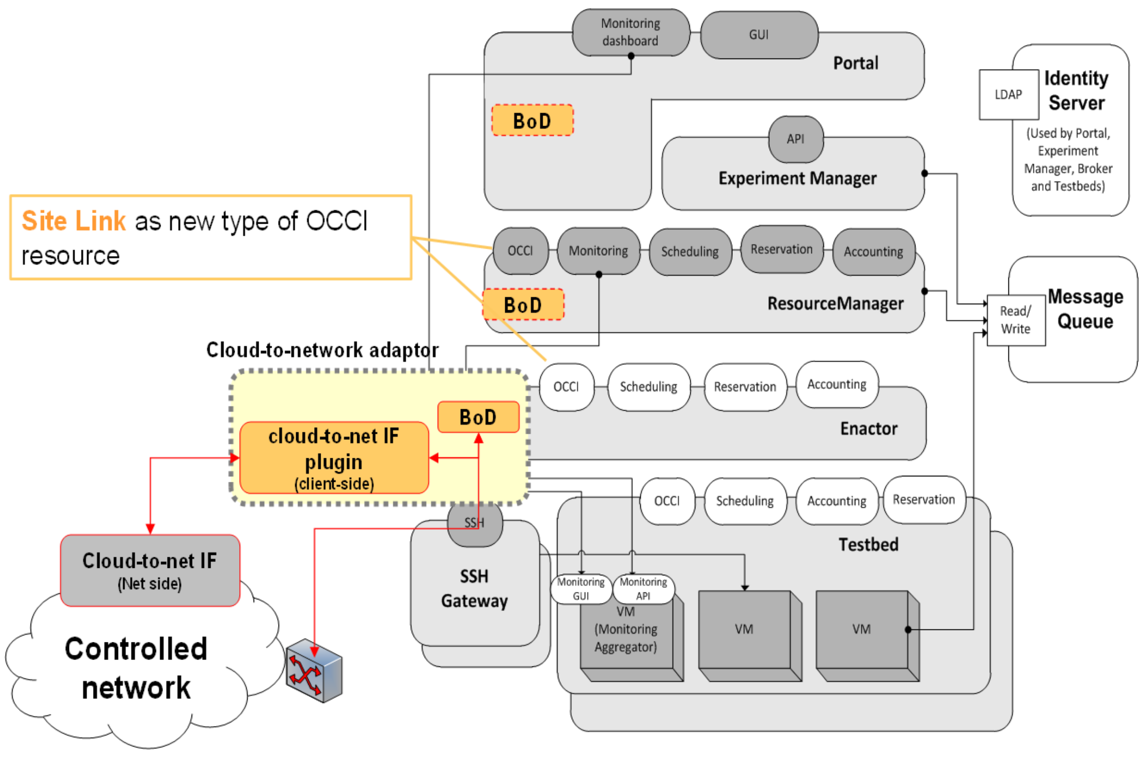

Figure 3.4: BonFIRE Architecture with Cloud-to-Net extensions

### 3.1.5 GridARS

GridARS[21] is a reference implementation of the Open Grid Forum (OGF) Network Services Interfaces (NSI), Connection Service (CS) protocol standard, developed by AIST. The CS protocol version 2 is a Web services-based interface to reserve, provision, release and terminate a service, such as a end-to-end connection, via a two-phase commit protocol. GridARS can coordinate multiple resources (services), such as a network connection, virtual machines and storage spaces, via the CS protocol in order to provide requesters with a virtual infrastructure, which spans several cloud resources, provided by multiple management domains including commercial sectors.

Figure 3.5 shows a resource management configuration assumed by GridARS; Domain A and B denote network domains managed by different administrative organizations. GridARS modules could be configured in a coordinated hierarchical manner, or in parallel, where several resource coordinators denoted by GRS compete for resources with each other on behalf of their requesters, such as users and applications.

GridARS provides three service components:

- Resource Management Service (RMS)
- Distributed Monitoring Service (DMS)
- Resource Discovery Service (RDS)

Resource Management Service (RMS) is based on NSI CS and consists of Global Resource Coordinators (GRCs) and Resource Managers (RMs) for Computers (CRM), Networks (NRM), and Storage (SRM). Coordinating with GRCs and RMs, RMS enables to coordinate heterogeneous virtual resources on multiple cloud environment. GRC has a co-allocation planning capability, which determines a suitable resource allocation plan.

Distributed Monitoring Service (DMS) allows requesters to monitor the virtual environment allocated to them. DMS does not have a central database, but gathers distributed monitoring information, tracking the hierarchical RMS reservation tree using the reservation ID, automatically. DMS consists of Aggregators (DMS/A) and

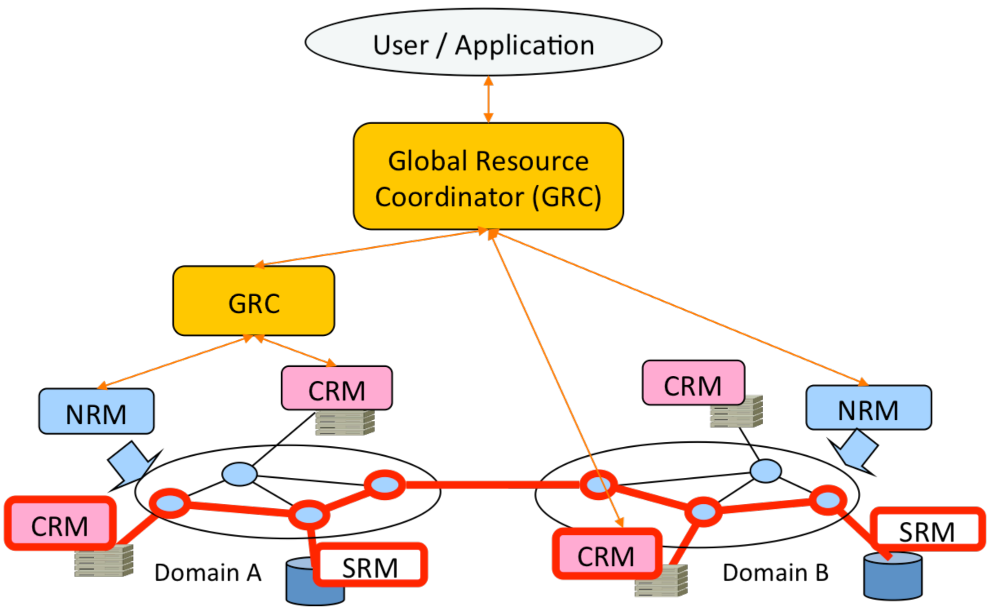

Figure 3.5: GridARS resource management configuration.

Collectors (DMS/C). DMS/A gathers monitoring information from related DMS/As or DMS/Cs distributed over multiple domains, and provides the information to the requester. Each DMS/C monitors the reserved resources periodically, filters the monitoring information by the domain policy, and provides the requester with the authorized information.

Resource Discovery Service (RDS) collects static resource information items from each resource domain and provides the aggregated information. The RDS implementation is based on Catalog Service Web (CSW), defined by Open Geospatial Consortium (OGC), which is an online XML-based database. Each resource domain can POST its static resource information, such as network topology, number of VMs, and storage spaces.

For further details, please refer to Appendix A.

### 3.1.6 RISE

Since 2009, JGN-X have been working on developing a nation-wide OpenFlow testbed: RISE (Research Infrastructure for large-Scale network Experiments). The RISE project is successfully running an OpenFlow testbed over JGN-X, fully utilizing its wide-area coverage from US West coast to Southeast Asia. RISE provides a wide-area OpenFlow network composed of hardware OpenFlow switches. It also provides the RISE OpenFlow Controller based on Trema and developed by NEC. Also researchers and developers can try their own OF controller on the RISE network for their experiment. SDN or Cloud developers can also try their own software experiments with the dynamic network provisioned by the RISE controller. Currently, RISE has 11 sites in Japan, and three sites overseas (see Figure 3.6). For each site, RISE has a number of OpenFlow switches and two VM servers (Japan domestic only). Currently, there is no control framework or portal. As such, the FELIX control framework will be a great contribution to the existing testbed.

**New RISE architecture for data-plane**

Through the operation of RISE testbed, the RISE team encountered the following scalability issues:

- Unevenly distributed OpenFlow switches: Administrators cannot assign OpenFlow switches for users to fit their required topology. Users concentrate in sites which have multiple links. For example, if user requests four OpenFlow switches with a loop topology, they can only be hosted in one of four sites.

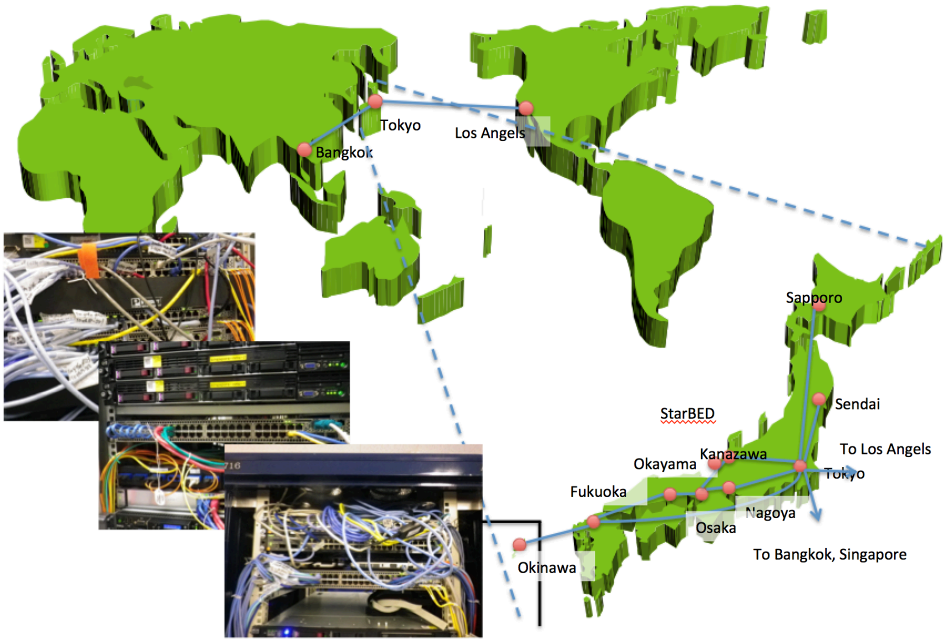

Figure 3.6: Global RISE testbed infrastructure

- Number of users: A multi-user environment is implemented using VSI (Virtual Switch Instance). The maximum users supported by this is 16 and therefore, only 16 users can share single physical OpenFlow switch at a maximum.

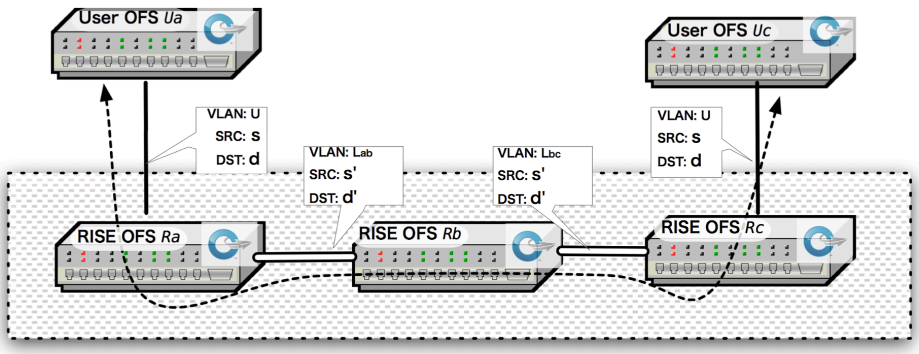

Figure 3.7: Implementation by rewriting MAC addresses

Currently, work is focusing on remedying the inability to freely create topologies using the given OpenFlow switches, particularly as this can constrain a user's experiments. To solve this issue, they design and discuss topology management system called "RISE3.0", which uses OpenFlow. In RISE 3.0, a "topology-separation function" is implemented which separates physical links and neighbour links between OpenFlow switches. This provides a more flexible experimental network. In order to realize the "topology-separation function", they introduced the concept of a logical path with MAC address rewriting. Actually, in the RISE controller, a logical path is defined by set of physical links. MAC addresses are then rewritten to guarantee the uniqueness of the packet in RISE OFSes

(Figure 3.7). The controller employs the address rewriting method to forward packets. It identifies flows with only the MAC address and VLAN ID (user ID). With this method, the OpenFlow switch can utilize the hardware processing path to forward packets without impacting performance. For further details, please refer to Appendix A.

## 3.2 FELIX Considerations

The table below summarizes the architectural elements giving a brief description of their objectives and functions. Moreover, the *FELIX Considerations* column is introduced for a future reuse of the element in the FELIX software framework.

| **Architecture Element** | **Brief Description** | **FELIX Considerations** |
|---|---|---|
| *OFELIA* | | |
| Expedient | Expedient is a pluggable, centralized GENI framework Graphical User Interface. It works on top of the OFELIA Control Framework and therefore allows to:<br>• to access OCF functions: setup, (de)allocation and monitoring of the experiment resources within the OFELIA facility<br>• to connect with Aggregate Managers through related plug-ins to display, provision and configure physical or virtual resources<br>• the experimenter to manage projects, slices and permissions<br>• the admin to manage users, add aggregates and approve project creation requests<br>Technical: web application developed with Python and Django; requires Apache2 | FELIX could use Expedient for its extendable architecture. It should enhance the module with new plug-ins for the new FELIX resources (e.g. inter-datacenters aggregate manager). |
| Aggregate Managers | This layer includes entities for the management and aggregation of homogeneous resources:<br>• VT-AM: for the management of virtual machines on Xen-based servers<br>• Opt-in manager/FOAM: for the management of OpenFlow-based switches<br>Technical: need XenServer and FlowVisor RMs, respectively | FELIX could reuse VT-AM and Optin-Manager to virtualize compute and network resources taking advantage of the underlying technology (XenServer and FlowVisor). FELIX should extend this layer with new aggregates (e.g. inter-datacenters connectivity aggregate manager). |

| | | |
|---|---|---|
| Resource Managers | Resource Managers manage physical resources (servers, OpenFlow switches, optical devices) using dedicated interfaces:<br>• FlowVisor is a transparent proxy between OpenFlow switches and multiple OpenFlow controllers. It is used to create "slices" of network resources enforcing isolation between each slice.<br>• XenServer is an open source virtualization platform for server virtual infrastructure. | FELIX could reuse the resource managers tools for their high quality and stability. Those seem to be completely in scope with the FELIX architecture. |
| *FIBRE* | | |
| MySlice | MySlice is a resource management tool used to list, filter, and reserve resources made available through the SFA control framework. It is a web framework based on a independent plugins and shared messages interface. The plugins provide operations for query editing, data display and resource allocation. | FELIX must provide a graphical user interface (GUI or WUI). MySlice is a good candidate to became the FELIX portal for its strong relationship with SFA specifications. Its SFA-registry could be used for AA (or AAA) services. |
| Expedient | Similar to OFELIA. | See OFELIA section (Expedient). |
| Aggregate Managers | Similar to OFELIA but with some additions, such as a common northbound SFA interface for the following:<br>• VT-AM: for the management of virtual machines on Xen-based servers (being adapted to AMsoil)<br>• Opt-in manager: for the management of OpenFlow-based switches<br>And also another aggregate:<br>• Roadm-AM: for the management of the Roadm optical devices (based on AMsoil)<br>Technical: need XenServer, FlowVisor and OpenNaaS RMs, respectively | See OFELIA section (Aggregate Managers). |
| Resource Managers | Similar to OFELIA, but including:<br>• OpenNaaS is an open source platform for the provisioning of network resources. It provides services for the deployment and automated configuration of network infrastructures. | See OFELIA section (Resource Managers). |

| | | |
|---|---|---|
| Enhanced Nox-controller | Enhanced Nox-controller is an enhancement of the NOX-classic OpenFlow controller used for network discovery and OpenFlow-based switches management. It can install/remove flow-entries in the underlying switches and retrieve OpenFlow monitoring information. It communicates with F-PCE module for a path computation inside a physical or virtual topology. It is based on the "old" Zaku branch of the NOX-classic. | FELIX should use one of the latest SDN controllers (e.g. OpenDaylight) to manage OpenFlow switches. The NOX-classic (old zaku branch) is now deprecated and is recommended to use POX (python version) or the new version of NOX (only C++ code). |
| Flow-aware Path Computation Engine | F-PCE is based on the IETF PCE architecture [16]. The module is responsible for the composition of the network services related to the end-to-end flow routes. It is a centralized entity providing algorithms for the path computation including the involved OpenFlow-based switches and the related flow-entries needed for the circuit activation. The code is closed by a NXW proprietary license. | FELIX should use a single path computation entity for setting up SDN slices and for connections between SDN islands. FELIX should extend F-PCE with the “slices” information and the real-time resource utilization (intra and inter data-centers). |
| *FED4FIRE* | | |
| Portal | Entry point that allows to register and check the discovered available resources, as well as allocation and provisioning. | See FIBRE section (MySlice). |
| Identity provider | Global registry of the federated users. May be overridden by a local instance. | FELIX could benefit from inspiring or adapting the local-if-not-global schema for the provider/registry. |
| Testbed directory | Lists info for all the federated testbeds, whether readable for humans (website) or computers (list of URIs). | FELIX could add this feature to improve the information offered, either for provisioning or monitoring purposes. |
| Tool directory | Gives an overview of available tools for the experimenters (website, through portal). | FELIX could add this feature to improve the information offered to its experimenters. |
| Certificate directory | Exposes a centralized repository of the trusted root certificates for the federated facilities. | FELIX should check if this approach is more suitable to register the federation certificates, rather than the SFA Registry/DB. |
| Future reservation broker | Facilitates future reservations of resources by finding and allocating the right time slots and resources over multiple testbeds. | FELIX should not integrate this functionalities inside the portal. Check instead whether to implement life-cycle checks and orchestration on a dedicated component or to replicate this in the hierarchical ROs. |
| Monitoring tools | Measure data for facilities, infrastructures and experiments. | FELIX could inspire from this to implement monitoring at different levels for both infrastructures, experiments and resources |

| | | |
|---|---|---|
| OMNI | Omni is a GENI command line tool for reserving resources at GENI Aggregate Managers (AMs) via the GENI AM API. 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. | FELIX could use OMNI as CLI tool in case it decides to use GENI as a communication standard. |
| SFI | SFI is another command line client for SFA interfaces for Discovery, reservation, provisioning and releasing and slice management: create, open, update, start, stop. 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 | FELIX could use SFI as CLI tool in case it decides to use SFA as a communication standard. |
| *BonFIRE* | | |
| Portal | The Portal provides a web graphical interface to access BonFIRE services. It allows to:<br>• visualize capabilities and resource availability for the different testbed sites.<br>• specify experiments through wizard procedures, upload of experiment descriptor or manual resource declaration.<br>• monitor the status and history of running experiments. | The Portal is strictly focused on experiments involving mainly IT resources, with simple networking scenarios. FELIX requires a more complex network virtualization and management, leading to a different information model, e.g. considering network slices. |
| Experiment Manager | The Experiment Manager schedules, plans and orchestrates experiments execution as specified in the experiment descriptors. The Experiment Manager provides an API to the Portal or other user agents, and manages the resources associated to an experiment through the Resource Manager. | FELIX architecture does not include a dedicated component for the management of experiments life-cycle and orchestration, but integrates these functionalities in the portal and the resource orchestrator. |

| | | |
|---|---|---|
| Resource Manager | The Resource Manager provides a resource level interface to create, update and destroy virtual resources (compute, network and storage resources), in correlation with experiments' life-cycle. The Resource Manager operates on the overall physical infrastructure and offers 3 facilities for on-demand provisioning and in-advance reservation, resource sharing and monitoring of resource usage. | The Resource Manager is a centralized entity that does not fit well to a distributed and federated environments. Moreover, the adopted information model is not fully compliant with the proposed FELIX model, due to the lack of concepts related to network virtualization (e.g. network slices). |
| Enactor | The Enactor abstracts the details of the different testbed implementations, providing a unified OCCI interface towards the Resource Manager. It includes adaptors to interact with the different cloud testbeds (e.g. Amazon EC2, GÉANT AutoBAHN Bandwidth on Demand service, FEDERICA infrastructure and OpenStack-based cloud testbeds). | Due to the distributed nature of the Aggregate Managers, the Enactor functionalities are not fully required in FELIX. Moreover, the Enactor exposes a north-bound interface based on OCCI extensions which are not compliant to SFA specifications. |
| Identity Manager | The Identity Manager is used to authenticate users and store their SSH keys. It is a LDAP server. | The Identity Manager functions will be covered by the SFA-registry or an LDAP database in FELIX portal. |
| Message Queue Server | The Message Queue Server provides publish/subscribe mechanisms for the other architecture components to exchange asynchronous messages and events related to infrastructure management and experiments. | The exchange of messages in FELIX architecture is still under discussion. |
| Collection Cache | The Collection Cache listens to events from all sites to keep track of the current state of each resource. | In FELIX architecture, these functionalities will be embedded in the Resource Manager. |
| Scheduler | The Scheduler implements the in-advance reservation system, keeping track of the time-slots reserved for the different resources. | Advance reservation functions have not been deeply discussed in FELIX architecture. |
| Accounting Service | The accounting service keeps track of the lifetime of resources used by the experimenters, generating the data that would be required by a billing system. | Accounting functions have not been deeply discussed in FELIX architecture. |
| Authorization Service | The Authorization Service is used to ensure that experimenters do not overuse their allocated quote of resources. | Authorization mechanisms in FELIX are managed through SFA procedures. |
| Monitoring Aggregator | The Monitoring Aggregator provides monitoring information at the application, virtual machine and infrastructure level. It is based on the Zabbix open source software. | The applicability of a Zabbix-based solution can be evaluated for the FELIX architecture. However, dedicated monitoring agents should be implemented for networking services. |

| | | |
|---|---|---|
| Elasticity Engine | The Elasticity Engine supports the elasticity-as-a-service, where VMs creation, modification and deletion is automatically managed depending on the current load. It is based on Zabbix as source of monitoring information and HAProxy/Kamalio for load balancers. | Elasticity functions have not been deeply discussed in FELIX architecture. |
| CoCoMa | The CoCoMa framework allows experimenters to emulate controlled operational conditions for contentiousness and maliciousness in shared environments. | This feature is not completely in scope with the FELIX project. |
| Experiment Data Manager (EDM) for Provenance | The EDM for Provenance is a tool to capture provenance data for experimenters and reasoning about this information in an ontological sense. | This feature is not completely in scope with the FELIX project. |
| *GridARS* | | |
| Global Resource Coordinator | GRC manages and coordinates heterogeneous resources. It could be configured to work in different manners (i.e. hierarchical or parallel modes) allowing the interoperability of zones managed by different GRCs. | FELIX could use GRC for coordination of Transit Network Resource Managers. |
| Resource Managers | They directly manage local resources. The project defines 3 RMs:<br>• Networks Resource Manager (NRM)<br>• Computers Resource Manager (CRM)<br>• Storage Resource Manager (SRM) | FELIX could use RMs as a wrapper of actual resource management systems, e.g., OpenStack, CloudStack and SDN controllers. |
| Distributed Monitoring Service | DMS monitors the virtual environment. It is a distributed system composed by Aggregators and Collectors.<br>• DMS/A gathers information from other distributed DMS/As or DMS/Cs.<br>• DMS/C monitors, filters and provides the reserved resources to the requesters. | FELIX could use DMS to collect distributed monitoring information. |
| Resource Discovery Service | RDS collects static resource information items from each resource domain, and provides the aggregated information. | FELIX could use RDS as a resource discovery service of computer and network resources. |
| *RISE* | | |

| | | |
|---|---|---|
| OpenFlow Controller | OF Controller is based on Trema, developed by NEC. It provides a Logical Path System which is a virtualization framework able to translate the physical OF switches into logical entities for traffic isolation. RISE supports 2 different methods: • VLAN stacking • Address rewriting | FELIX could use the OpenFlow Controller as base for the SDN Resource Manager, basically for the OpenFlow-enabled physical switches. |

Table 3.1: Architectural Elements of Existing Testbeds

## 3.3 Summary

In this section, we briefly summarize the advantaged of the testbeds analyzed within this section, focusing on what is missing (or seems to be not fully covered) in their proposed architectures and offered services.

Our analysis takes into account different modules, services and interfaces. For instance, in the **OFELIA**-based testbeds there is no concept of an Orchestrator, which might be seen as a drawback in large federations. Moreover, OFELIA testbeds are inspired by SFA-like architectures where the federation occurs at the resource level, meaning that the clients have direct access to the different Resource and Aggregate Managers. This, again, can be viewed as a major problem for scalability within distributed architectures.

Other testbeds such as **BonFIRE** are mainly focused on cloud computing, therefore giving priority to computing rather than network resources. This results in a lack of a slice concept becomes complex to federate with. However, it can provide dynamic network parameter configuration (i.e. latency) and in release 3.1, it offers bandwidth on demand services by using GÉANT's AutoBAHN as a third party interconnect between the EPCC and PSNC sites (using NSI).

Meanwhile, **FED4FIRE** is an ambitious federation of heterogeneous testbeds, but it is still mainly focused on how to effectively offer the different services of the testbeds, and for the most part, avoids the matter of how to serve to the clients/users all the federated resources as a single logical plane. However, this is one focus of current efforts within the project. Furthermore, the network connections between testbeds is fixed and cannot be manipulated. This results in a lack of BoD.

Table 3.2 gives a schematic overview of our analysis.

| **Project** | **Advantage** | **Disadvantage** |
|---|---|---|
| OFELIA | The LDAP, as a single and centralized element, and its credential management logic (currently within Expedient) could be extended and integrated in the FELIX framework. There is a design for a Clearinghouse component that aims to modularize and externalize the credentials management that could be completed and used in FELIX. The Control Framework Layer (SFA-based) could be enhanced to manage the different FELIX SDN islands. | The inter-domain (inter-testbeds) network segment seems to be not fully integrated in the developed architecture. FELIX could cover this gap with its new architectural element (NSI Resource Manager). Moreover, FELIX could orchestrate the IT & networking resources with an higher-level layer (Resource Orchestrator). |

| | | |
|---|---|---|
| FIBRE | The SFA specifications are adopted to manage and federate testbeds. They could be enhanced for the different FELIX SDN islands. | The inter-connections between different testbeds are based on VPN (or other L2) technology. FELIX should provide a programmable architectural element for the inter-domain network segments (Transit Network Resource Manager). A dedicated architectural element to orchestrate resources seems to be missing. FELIX should provide a component for this purpose (Resource Orchestrator). |
| FED4FIRE | The “broker” concept and its functionalities could be extended to allow an easy discovering phase of the FELIX Resource Managers and Orchestrator. | The interconnections between different testbeds is covered by a dedicated component that seems to be not NSI-based. Moreover, there is not Bandwidth-on-Demand functionalities planned. Problems such as scalability have yet to be analized. |
| BonFIRE | The network system, based on GÉANT BoD service (AutoBAHN), covers the inter-testbeds connections. FELIX architecture could adopt an architectural element (Transit Network Resource Manager) to provide a Bandwidth-on-Demand services over a network domains. | The federation concept is not investigated and the interfaces are not SFA compliant. |
| GridARS & RISE | The NSI protocol and the generic developed framework could be introduced in the FELIX architecture providing a new architectural component able to manage the inter-domain network segment (Transit Network Resource Manager). | The projects seem to be SFA agnostic. Currently, there is no control framework or portal. FELIX could provide a dedicated control framework to manage their OpenFlow resources. |

Table 3.2: Overview of advantages and disadvantages of alternative approaches

# 4 System Architecture

FELIX aims to provide a framework which integrates resources of different kinds (e.g. transit network, SDN, IT, etc.) from a multidomain heterogeneous environment.

In order to implement such functionality, we need a flexible and scalable architecture which can assure a sufficient level of interaction between various system components. This section introduces the core building blocks of the FELIX architecture. Note that the detailed specification of the mechanisms, protocols, and interfaces which will implement the user requests over the federated testbed infrastructure are not in the scope of this deliverable. FELIX has opted to use a hierarchical model for inter-domain dependency management, with orchestrator entities responsible for synchronization of resources. The overall architecture is illustrated in Figure 4.1.

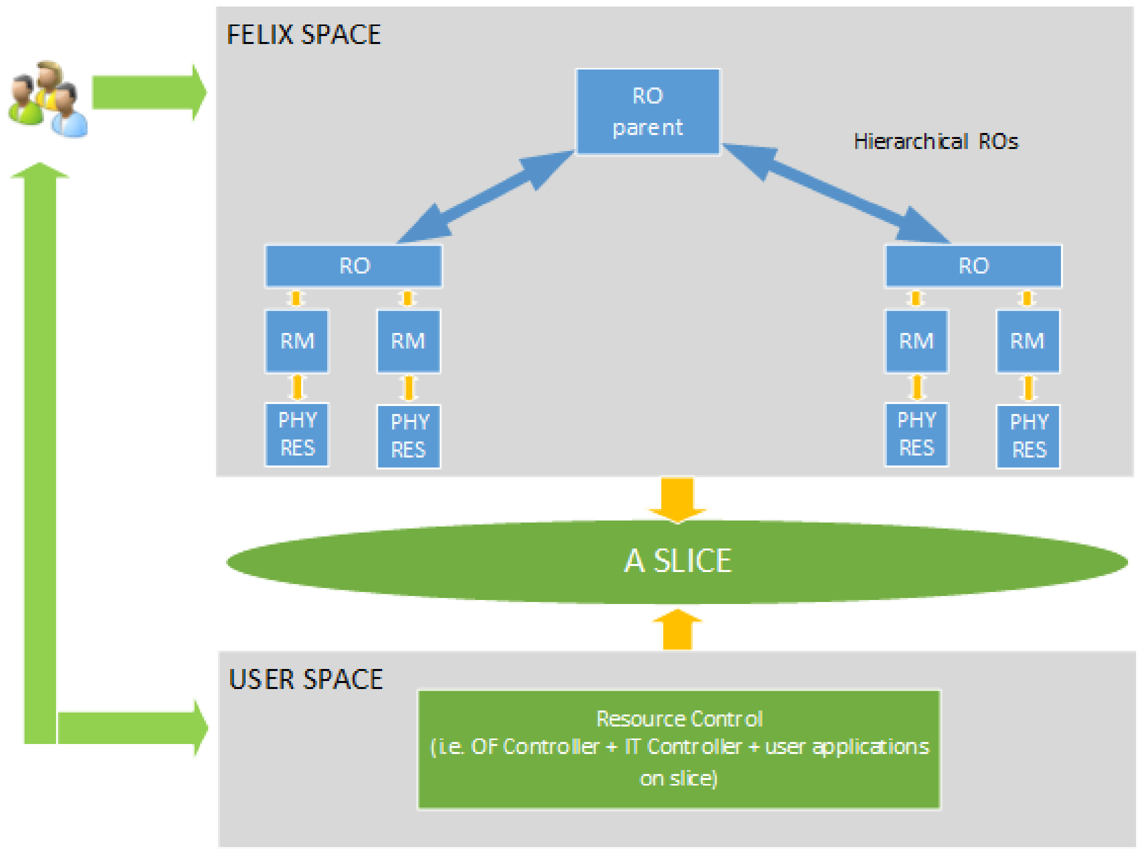

Figure 4.1: FELIX overall architecture -- ROs use RM to manage physiacal resources of different types (e.g. Transit Network, SDN, Computing Resources).

The architecture can be seen as the combination of two spaces: the FELIX Space and the User Space. Both spaces play distinct roles in the creation and operation of each and every slice, which is created by the FELIX Space upon a user request, and further on managed by user tools in User Space.

**FELIX Space** The FELIX Space relies on Resource Orchestrators (ROs), Resource Managers (RMs), and the physical (testbed) infrastructure to provide the resources needed for realizing a user slice. Resource Orchestrators (ROs) are stateful entities, which receive user requests and can either serve them directly or determine which other RO or Resource Manager (RM) can serve the request. In effect, an RO can decide to delegate whole or part of a user request to another RO or a specific RM. Each RM typically manages a specialized type of resources. For example, a RM may oversee the exact network resources in an SDN island, port interconnection across islands, as well as computing and storage resources within a single administrative domain. We expect that each domain may have one RO and several RMs, although this is an administrative and operational policy issue primarily. Similarly, all RMs within a single domain should have single RO supervising them and be located at the bottom of the RO hierarchy. Note that an RO may not have RMs associated with it. An RO may act as a resource aggregation agent and can thus represent multiple administrative domains as one resources set to FELIX users.

In relation to single administrative domain, a RM can operate in single domain only (see Figure 4.1). It can however perform inter-domain operation, e.g. by peering with RM in adjecent domains. All RMs in single domain must be supervised by single RO, which is as well single domain scoped, yet it requires the ability to perfom inter-domain operations. The ROs which has no direct RM associations but controls one or more other ROs in a tree model (see RO Parent on Figure 4.1), are considered to be inter-domain entities, as their range of responsibility is not linked with single domain only.

The proposed architecture aims to scale easily: one can always add new RO(s) and/or new RMs under any RO, as needs arise. Further, the mechanisms proposed in this document will enable new installations to easily adopt and extend whole FELIX infrastructure. The preferred way of communication between ROs is a tree model, where ROs follow the defined hierarchy in order to contact each other. We foresee that horizontal communication will be limited. Optionally, however, the architecture permits a peering model, where ROs can contact each other directly, without the need of inter-mediation. The choice of communication manner is up to the implementation and the practical needs of specific deployments of the FELIX architecture.

In the FELIX architecture, a user of the federated testbed infrastructure always sends the requests to an RO (typically using a graphical user interface or through the use of a programmatic interface) and does not communicate directly with RMs of any domain. This unique entry point into the RO hierarchy guarantees that the request will be processed (and delegated as needed) by the most appropriate ROs and RMs. This design approach aims to ensure that the valid administrative domains and resource pools will be designated to implement a user slice. As each user slice may comprised of a set of resources which are of different type, multiple RMs, either in a single or multiple domains, may be involved in realization of user request.

**User Space** The User Space consists of any tools and applications that a user wants to deploy to control a slice or execute particular operations within it. The selection of tools is completely dependent on the user requirements and his/her decisions and is out of the FELIX management framework scope. Each user slice defines the boundaries of the user-controlled environment. The slice also guarantees isolation of this environment from the physical resources and other user slices. As many use cases scenarios discussed in [FELIXD2.1] require SDN resources to be committed into a slice, an example of a user tool may be an OpenFlow controller, which will enable users to configure SDN flows within a slice according to their needs. Another example is, say, an IT Controller, which can provide remote access to servers, clusters or data storage facilities included in a slice.

The following sections explain in more detail the functionality and business logic of the particular components of the architecture in both the FELIX and User Spaces, and their dependencies.

## 4.1 Concepts and Definitions

This section revisits the concepts and definitions introduced in [20] as we proceed in describing the overall FELIX architecture and the core functional building blocks.

We briefly introduce the main concepts related to the physical distributed infrastructure of the FELIX facility which is organized into multiple administrative domains. Basically, an **Administrative Domain** is a service security provider that holds security repositories and authenticates and authorizes clients with credentials safely and easily. In the FELIX infrastructure, every administrative domain could be controlled and managed by different architectures and interfaces: Software Defined Networking (SDN) [17], [11] and Transit Network Service (TNS) such as Network Services Interface Connection Service (NSI-CS)[22], [8]. The "*FELIX physical network infrastructure concepts*" subsection is consequently focused on these technologies.

In the FELIX vision, the interconnected experimental facilities are federated, so that multiple virtual infrastructures spanning across several domains can be delivered to the user as isolated set of integrated resources.

The FELIX architecture adopts Slice-based Federation (SF) to fulfil this requirement. The "*Slice-based Federation concepts*" subsection presents the key concepts related to slice-based federation and some details about the architectural components which will be imported and enhanced in the FELIX architecture.

Finally, the "*FELIX architecture definitions*" subsection provides the terminology adopted to identify the main elements of the FELIX architecture, which are conceptually based on the previous concepts and definitions (i.e SDN, TNS, SF). This subsection could be used as reference when reading the detailed description of each single component in the next chapters.

### 4.1.1 FELIX physical network infrastructure concepts

FELIX infrastructure includes distributed facilities able to manage and control computing/storage (**CR**) and networking (**NR**) resources. Each network facility can be organized as a set of SDN-controlled network domains. The different experimental facilities are interconnected through TNS-controlled network domains with the scope to enable the inter-domain connectivity.

**Software Defined Networking (SDN)** is a relevant new term for the programmable networks paradigm, as discussed in [17]. In short, SDN refers to the ability to use software to program individual network devices dynamically and therefore control the behaviour of the network as a whole [15]. The separation of the network control plane, which oversees several devices, from the forwarding plane (the data plane) serves as a foundation for a dynamic, easily manageable, cost-effective, and adaptable architecture. As the network control becomes directly programmable, the underlying infrastructure can be abstracted for applications and network services, reducing the dependency on the manufacturer. In many cases, SDN is identified with the use of a particular protocol (e.g. OpenFlow) to allow the communication between the control plane and the data plane, but the truth of the matter is that SDN concepts are far more general. For the particular SDN definition advocated by the Open Networking Foundation please see [11].

The basic unit of the FELIX architecture is the concept of *resources* (**NR & CR**): network (switches, routers, optical devices..) and computational (physical server or VMs, storage, blocks, objects…) resources. The SDN mechanisms are adopted to organize the physical network resources in a variety of **SDN zones**. In FELIX terminology, a *SDN zone* is a set of resources grouped for homogeneity of technologies and/or control tools and/or interfaces (e.g. L2 switching zone, optical switching zone, OpenFlow protocol controlled zone or other transit domain zone with a control interface). The major goal of defining SDN zones is to implement appropriate policies for the availability, scalability and control of different resources in a **SDN island**. An *SDN island* is defined as a set of virtualized NR and CR under the same administrative ownership or control (an *administrative domain*).

The different SDN islands can be grouped in the *Future Internet (FI) experimental facilities* (or **SDN-controlled network domains**). The FI experimental facilities are controlled by dedicated software, which exposes interfaces which can be used by a federation framework to orchestrate resources in a multi-domain environment.

Finally, the overall FELIX infrastructure is composed by several (distributed and federated) *FI experimental facilities* physically interconnected using **TNS-controlled domains**.

In FELIX architecture, the network domain (or network domains) which connects the distributed experimental facilities will use the **Transit Network Service (TNS)** to offer the automated and on-demand control of the connectivity services and, optionally, enable inter-domain topology exchanges. NSI-CS is one of the TNS which can be used in the FELIX implementation. The NSI is under standardization within the Open Grid Forum (OGF) NSI Working Group, which has also defined a general framework to deliver network infrastructures as a service and a Connection Service protocol to enable the automated creation of network circuits in multiple and heterogeneous domains. Moreover, the NSI framework supports the federation concepts, as fully described in the "*Transit Network Resource Manager*" section of this document. In FELIX, the Transit Network Service protocol will be used to orchestrate the resources in the TNS-controlled network domain and establish inter-domains connectivity with a specific granularity. This approach allows to create experiments spanning resources from different domains and continents and requires the deployment of an Agents of TNS within islands and TNS-controlled network domain.

The Figure 4.2 gives a graphical representation of the FELIX network infrastructure concepts.

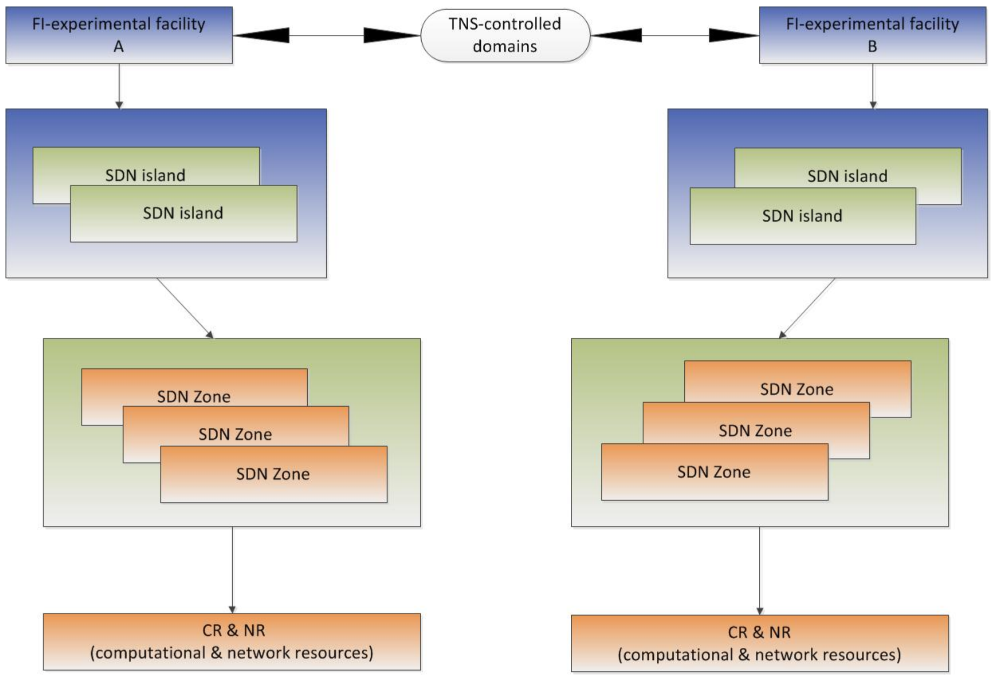
FI-experimental facility A
TNS-controlled domains
FI-experimental facility B
SDN island
SDN island
SDN island
SDN island
SDN Zone
SDN Zone
SDN Zone
SDN Zone
SDN Zone
SDN Zone
CR & NR (computational & network resources)
CR & NR (computational & network resources)

Figure 4.2: FELIX physical network infrastructure concepts

### 4.1.2 Slice-based Federation (SF) concepts

In FELIX, the administrative domains corresponding to the distributed infrastructures are federated to compose multiple and isolated **FI experimental facilities**. SFA and NSI Framework are among the architectures which can support FELIX system requirements.

SFA v2.0 specification [10] is one of the SF protocol and interface which can be used in the FELIX implementation. SFA v2.0 defines a control framework architecture to allow a federation of slice-based network substrates to interoperate. In this context, SFA identifies two authority roles for the control and management of a federated system:

- **Management Authority (MA)**: responsible for a subset of physical components and ensures the proper behaviour of the components (that is, that hose execute the resource allocation accordingly)
- **Slice Authority (SA)**: responsible for the registration and control of one or more slices as well as managing the user access to the slices

In FELIX, SFA can be used to provide a federation framework between the existing testbed management platforms deployed at partners' premises.

The main concepts in the SFA framework are summarized in Table 4.1.

| **SFA key concept** | **Description** |
|---|---|
| Resource | Resources include physical resources (e.g., CPU, memory, disk, bandwidth), logical resources (e.g., file descriptors, port numbers), or synthetic resources (e.g., packet forwarding fast paths). Resources are described through a resource specification (RSpec), typically expressed in XML format following specific schemas. RSpecs are used to list (advertisement RSpec), reserve (request RSpecs), or describe reserved resources (manifest RSpecs) [5], [4] |

| | |
|---|---|
| Component | Components are the primary building blocks of the SFA architecture (e.g., an edge computer, a customizable router, a programmable access-point, etc..). Every component can encapsulate a set of homogeneous or heterogeneous resources, depending on the nature of the component. |
| Aggregate | An Aggregate is a set of components which are under the authority of the same MA, which also governs the aggregate. |
| Aggregate Manager | An Aggregate Manager is a logical element which controls and manages an aggregate. If the aggregate contains a single component, the Aggregate Manager could be also called Component Manager. |
| Sliver | A sliver can be considered as a resource container, which guarantees the isolation from every other sliver belonging to the same component. This requirement can be fulfilled via component virtualization or partitioning the component into distinct resource sets. Either way, the user is granted a sliver of the component. |
| Slice | A user-defined subset of virtual networking and computing resources, created from the physical resources available in federated testbeds. A slice has the basic property of being isolated from other slices defined over the same physical resources, and being dynamically extensible across multiple testbeds. On top of each slice, a specific set of control tools can be instantiated, depending on the specific domains it traverses. |

Table 4.1: SFA main concepts

The **slice** concept is adopted in FELIX with reference to the experimental facilities to be provided on top of the FELIX physical infrastructure. All the FI experimental facilities will be controlled programmatically through well-defined interfaces by the Slice-based federation framework, which orchestrates resources in a multi-domain environment. These facilities, extending the slice concept in SFA, are composed of compute and network resources (*CR* and *NR*) belonging to distributed SDN islands in FELIX infrastructure, interconnected via TNS-controlled domains (Figure 4.3). Moreover, a slice can collect resources of different types from several SDN zones within a single SDN island.

The SFA RSpec used in the requests is the native RSpec format of PlanetLab AMs. As of January 2012, PlanetLab supports GENI v3 RSpecs, which is the recommended RSpec to use in GENI, as [5], [4].

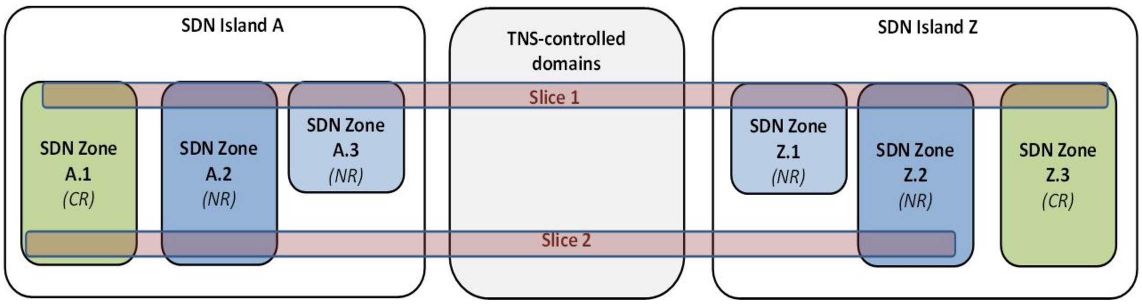

Figure 4.3: FELIX infrastructure key concepts

A fundamental element defined in the SFA framework is the **Aggregate Manager** (or Component Manager) which exports a well-defined and (remotely) accessible interface. FELIX architecture adopts and extends the SFA concept of Aggregate Manager with the introduction of architectural elements able to manage and control the FELIX-specific resources, in particular SDN and TNS resources (*NR*). In the FELIX terminology, these managers will be (generically) called **Resource Managers** and will expose well-defined interfaces for resource discovering and reservation. An example of SFA-based interface is the GENI Aggregate Manager API [4] which provides standardized reservation mechanisms. This interface is directly derived from the SFA 2.0 standard and is currently implemented by several GENI projects (e.g., PlanetLab, ProtoGENI, ExoGENI, InstaGENI).

On the other hand, while the NSI CS (Connection Service) is an interface to request provisioning of a network connection, NSI Framework itself is a general framework to reserve resources in advance, and manage provisioning of the resources during the reserved period. In the architecture of the NSI, each provider's resource is managed by a Network Service Agent (NSA). The NSI is the service interface between NSAs. An NSA can take on the role of a requester, a provider, or both (an aggregator). Multiple NSAs form a recursive framework of requesters and providers. Requests can be propagated through this framework of NSAs using a tree or chain workflow. An aggregator NSA can aggregate resources from multiple children NSAs and provide resources to the requester. Aggregator NSA corresponds to RO in the FELIX architecture. The Figure 4.4 shows the overview of NSI Framework.

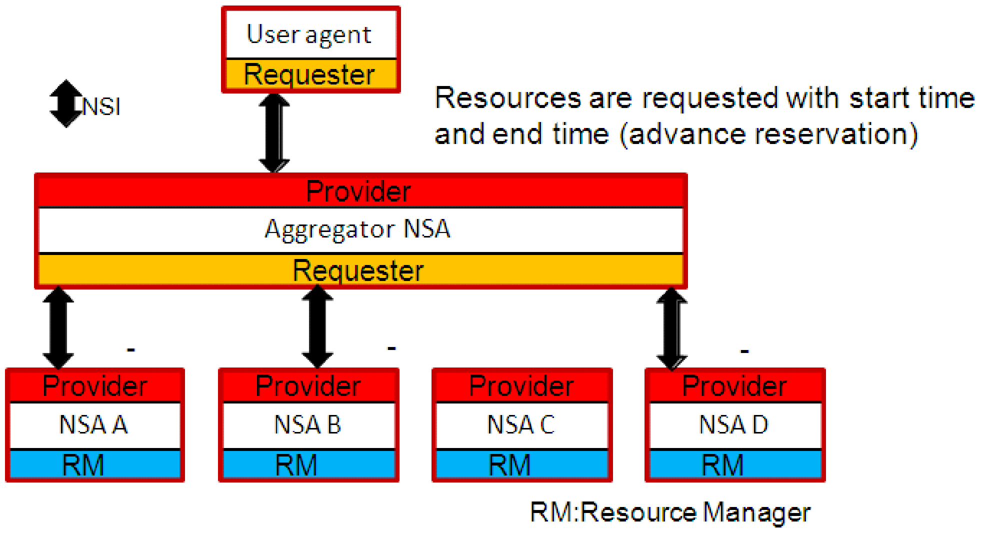

Figure 4.4: NSI Framework

NSI Framework supports advance reservation of resources. An NSI reservation is created using a two-phase commit process. In the first phase (reserve) the availability of the requested resources is checked; if the resources are available they are held. In the second phase (commit) the requester has the choice to either commit or abort the reservation that was held in the first phase. By this mechanism, requester can search available resources while holding some of resources which availability has already been confirmed. If a requester fails to commit a held reservation after a certain period of time, the provider may time out the reservation and the held resources can be released. Using this two-phase commit process, modification of a reservation is supported. During a reservation period, actual provisioning (activation) of resources can be controlled from the user. By sending a provision request, the resources are activated, and by sending a release request the resources are de-activated. De-activated resources can be re-activated by re-sending provision request.

In NSI, the concept of Service Definition (SD) is used to define characteristics of resources. In several SD have been defined for Connection Service in the NSI-CS protocol. By defining SD for computers and storages, those resources can be also managed by NSI Framework.

### 4.1.3 FELIX architecture definitions

This section provides a brief summary of the main architectural elements, which will be described in the next chapters. Some of these components (e.g. the Resource Managers) are conceptually derived from the SFA architecture and NSI Framework, while further new elements (e.g. the Resource Orchestrator) have been introduced to add innovative features like automated resource coordination and end-to-end service provisioning.

Table 4.2 gives an overview of the FELIX architectural components.

| **Architectural component** | **Description** |
|---|---|
| Resource Orchestrator (RO) | Architectural component responsible for the orchestration of the end-to-end, multi-domain service in the federated infrastructure. It coordinates the reservation and allocation of heterogeneous network and compute resources in each segment. Those resources are described through a common language, which can represent CR and NR characteristics and their constraints. |
| Resource Manager (RM) | Component in charge of controlling a specific type of resource, being the equivalent of the SFA Aggregate Manager and NSA in NSI Framework. FELIX defines two types of RMs: the TN RM and the SDN RM, to manage TN and SDN virtual resources respectively. In particular, the SDN RM will control the OpenFlow-based L2 resource in the access domains, while the TN RM will interact with the TNS-controlled domain for on-demand provisioning of connectivity in the transit segment. |
| Monitoring framework | Collects and manages monitoring data from infrastructure slices and experiments, including information retrieved from the different SDN islands and from the transit domains, in terms of performance of the established connectivity services. The monitoring framework is expected to interact with the Resource Orchestrator for providing aggregated measurements. |
| Slice Resource Controller | Element in charge of managing the creation, modification and deletion of slices related to the experiments (including all functionality, APIs and applications). |

Table 4.2: FELIX architectural components

## 4.2 Architectural Building Blocks

This chapter defines the roles, responsibilities and dependency of the FELIX framework components. It is divided into two sections, which focuses on FELIX and User Space and functions available in particular spaces. In general the FELIX Space responsibility is to manage resources and provide slice to the end users, while User Space tools allows end users to control their slice environment and execute their scenarios.

### 4.2.1 Management and Orchestration Architecture

The Management and Orchestration components instantiates the FELIX Space environment, providing end users and administrators with tools to manage the resources, maintain them and implement users' slices over them. This space consist of the following main components:

- Resource Orchestrator
- Resource Manager, i.e. Transit Network Resource Manager, SDN Resource Manager, and Computing Resources Manager
- Monitoring
- User Access/GUI

The components are explained in the following chapters.

#### 4.2.1.1 Resource Orchestrator

The **FELIX Resource Orchestrator (RO)** is a key functional element of the FELIX architecture, as a whole, and the centerpiece of the management and orchestration system design, in particular. In short, the FELIX Resource Orchestrator is responsible for orchestrating the end-to-end network service and resource reservations, possibly including relevant CR services, for the entire FELIX federated infrastructure, i.e. intra- and inter-testbed wise. We aim for an RO design that can coordinate end-to-end resource and service provisioning in a technology agnostic way. Resource provisioning ensures that all required intra- and inter-testbed resources are delivered to the requesting user for the specified period at particular locations.

We consider that the RO operates over a federated testbed infrastructure which consists of "SDN islands", such as the OpenFlow-based research testbeds in Europe, interconnected via what we refer to as "Transit Network Service (TNS) controlled domains", i.e. testbed interconnection facilities which are compatible with the Network Services Interface (NSI) architecture **[8]**. Each SDN island may have several SDN Zones. Given these core constituents, the FELIX federated infrastructure resources must be orchestrated in order to serve the use cases defined in [20]. In order to do so, we consider that an FELIX end-to-end service, e.g. a video stream, lives within a cross-island slice.

The key functions of the FELIX RO can be summarized as follows:

- The RO manages the different FI experimental facility user in terms of resource and data access policies.
- The RO mediates between the user (e.g. an experimenter wishing to employ the FELIX federated infrastructure via the use of a portal) and the technology-specific Resource Managers (RMs). We expect to have different RMs which will handle, for example, technology-dependent aspects in SDN domains and transit network domains (TN RM), as well as compute and storage resources (CR RM). As part of this mediation, the FELIX RO will be engaged in the creation (provisioning), maintenance, monitoring, and deletion (release) of the used resources and slices.
- The RO maintains a high-level (abstract), cross-island topological view, which summarizes the different (CR and NR) resources available along with their inter-connections. This topology view is initialized and updated by the underlying Resource Managers, thus implementing a distributed hierarchical resource discovery function.
- The RO determines which domains and which inter-domain resources should be used to instantiate a given end-to-end service for a FI experimental facility user's slice. For example, based on a user request for a given type of service to be instantiated in two remote islands, the FELIX RO determines which specific SDN Zones or resource domains should be involved.
- The RO coordinates and ensures that the correct sequence of actions takes place with respect to the operation the technology-specific Resource Managers. This includes the provisioning of the slice resources and as per the user requirements.
- The RO collects and correlates alarms on resources, on a per-slice basis, and proceeds with reporting/notifying the corresponding users on a per slice basis.

We note that the FELIX Resource Orchestrator operation is orthogonal to the datapath operation of different slices and the traffic of their respective FI experimental facility users.

The FELIX RO exposes all its functions through southbound and northbound interfaces, called FELIX Resource Interface, and inspired to the Slicebased Federation Architecture.

These interfaces allow us to manage resources from different experimental facilities in a common abstracted manner, using the resource description language **[9]** developed for the SFA **[10]** which can capture the characteristics of a given technology domain.

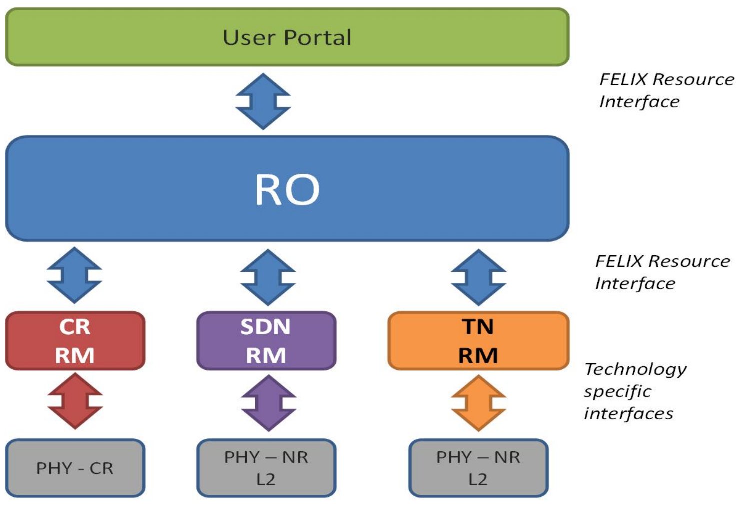

Figure 4.5: FELIX Resource Orchestrator positioning in the FELIX architecture.

With reference to a commonly accepted characterization of the IT service life cycle **[6]**, the following phases can be identified for a service that requires network and compute resources, as discussed in the use cases presented in [20]:

1. *Service Design and Planning*, during which the service characteristics, template resources, capacity and availability requirements are identified. In addition, this phase includes the detailed description of the relationships between the modules and components that will be involved ;

2. *Service Provisioning and Delivery*, during which the exact resources and correlations among them are put in place (delivered), the specific configuration items implemented and the overall service validated through testing before being accepted for the subsequent operation phase. Validation at this step may consist of status checks on resources (e.g. VMs up and running, query status on circuits or L2 data paths, etc.).

3. *Service Run-time Operation*, during which the service is active and monitored to react to any potential event based on its configured characteristics (dynamic updates, or more traditionally incidents, problems, requests). Monitoring is a key function in this phase since it allows to correlate service performances against a set of Service Level Agreements (SLA). Due to the different granularity of the monitoring data coming from different technologies in the data plane, an orchestration function is also needed to harmonize measures into a single monitoring framework and e.g. to react/orchestrate upon monitoring events like up/down events, thresholds crossing, etc. For example, in case of a L2 switching domain controlled via OpenFlow, the flow installation task with matching of a given classification tuple lives entirely in this operation phase as one of the key activities under the ownership of a SDN controller; similarly, a resource scaling up/down or migration from one island to another due to specific events like faults, time of the day, etc. are examples of run-time operations possibly automated and without extra delivery of resources.

4. *Service Termination*, during which the service is terminated and all associated resources are released in order to be available for other uses.

The resource orchestrator is involved in the following three phases: Service Design and Planning; Service Provisioning and Delivery; and Service termination. The FELIX RO does not play any major role in actual service operation.

The FELIX RO role in an end-to-end service life cycle is better illustrated via a walk through in an exemplary process flow. We start with the *Service Design and Planning* phase as follows. Assume that an experimenter (i.e. a user of the FELIX federated infrastructure) defines through the FELIX portal a cloud service specification. For example, the user could detail the type and version of the operating system and software applications to be used, the amount of storage required, the computing resources needed, as well as their inter-relations, such as which applications run on which resources. In addition to this, the user will need to describe the desired connectivity between the different components of the cloud service, i.e. define certain network parameters that are deemed critical (and let the FELIX architecture determine the remaining parameters). Finally, the user could specify the connectivity between the defined cloud service and the user. The cloud service description may include QoS parameters like bandwidth and delay, or service survivability requirements (with an impact on the recovery strategy to be adopted at the network level). Also, the user can specify automated elasticity rules that identify the modifications to be applied in the cloud and network service topology under certain conditions (e.g. what should be the reaction in case of changes in the traffic load or disasters).

The FELIX RO elaborates the requests received from the portal and determines the sequence of resource domains and the characteristics of the related resources which will be required to create the end-to-end service in terms of computing, storage, and connectivity services. During the *Service Provisioning and Delivery* phase the RO interacts with the technology-specific Resource Managers to request the allocation of the assorted resources in the FI experimental facilities. Therefore, each Resource Manager implements the decomposition and domain-specific configuration of the different abstract resources into specific resources of the SDN zone (again, this includes network, storage and compute resources) and of the transit network. In order to achieve this, each RM has its own decision entities (internal or delegated to a legacy technology function) to achieve better utilization of the whole infrastructure under control. Once the request is successfully concluded, the FELIX RO triggers the allocation and activation of the requested resources. This commit step is also mediated by the different involved FELIX RMs, and uses the technology-specific interfaces exposed by the different domains, like NSI for the transit domains, OpenFlow for the SDN domains, the API exposed by the Cloud Management System responsible for each involved data center for the CR domains.

In order to cope with the potential applicability of the FELIX architecture at large scale, the FELIX Resource Orchestrator is designed to be recursive, and seamlessly interface to a user portal or another parent RO. This mode of operation allows for establishing a hierarchy of ROs as illustrated in Figure 4.6. The parent RO coordinates the different child ROs (attached to their RMs) taking care of the overall supervision of the summarized network and CR topology exposed by the different ROs.

As Figure 4.6 illustrates, the user portal can alternatively decide to connect to any RO instance involved in the planned service (e.g. the RO for the source domain, or the RO for the destination domain, or even the parent RO).

In order to provide the possibility to transparently stack ROs hierarchically, it will be most useful to employ the same API on all ROs. Specifically, child-ROs shall not have a different northbound API than their parents. This recursiveness enables transparent changing of the underlying RO instances and enables the RO clients to interface with all ROs in the same way, no matter which granularity they encompass.

We expect that the mechanisms for optimizing the service and slice state consistency across the various ROs will be refined in further iterations of the FELIX architecture based on implementation and experimentation experiences as the project progresses.

Moreover, the FELIX RO should cooperate with the other FELIX architectural elements (e.g. GUI/Portal) to cover all the security aspects related to an Authentication Authorization Infrastructure (AAI). In short, it should confirm the user's identity (authentication) and associate the identity with rights and permissions (authorization). Through the FELIX framework, users can access services in a secure and confidential manner, simply by using e.g. their credentials or certificates. In other words and as a general requirement, FELIX (as a multi-user service) needs some mechanism to manage who can access the services and which actions each user can perform.

Orthogonal to the hierarchical design of the FELIX RO, all entities blend into a AAI (Authentication Authorisation Infrastructure). This architecture ensures, that all operations on the RO and its sub-entities are only performed by authorized actors. These actors can be administrative/operational personal, experimenters or auto-

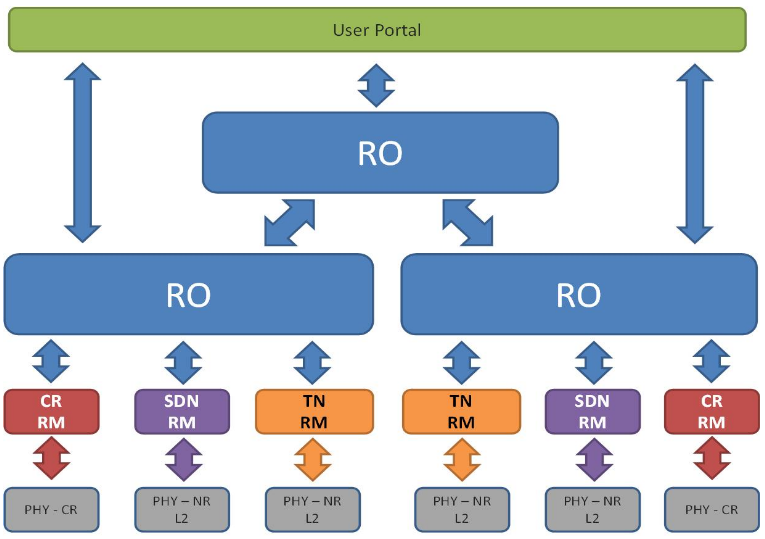

Figure 4.6: Hierarchical Resource Orchestrator in FELIX

mated actors. In order to secure the system against accidental misuse and malicious attackers, FELIX implements functions for authentication, authorization and accounting. For authentication, FELIX will use a certificate-based infrastructure and provide a CA (Certificate Authority) entity. Each entity in the FELIX architecture can choose to trust one or more CAs. When one of these entities needs authenticate a request, the certificate sent by the client can be verified against the CA's certificate. Only if the client's certificate was signed by a trusted CA, authentication succeeds.

After establishing, that the client is who it claims to be, the requested operations need to be authorized. In order to verify privileges of actors, the CA issues credentials. Credentials are signed documents which contain a mapping from the signing entity to an actor. Associated to this mapping is a set of roles which specify which operations may be performed by the targeted actor. To enable entities to delegate their privileges to sub-entities (e.g. to perform actions which are performed by lower layers in the hierarchy), credentials can also be chained. Usually the CA issues certificates to trusted members/actors, thus building the start of a trust chain. In order to understand the chaining, let's consider the following example: The CA trusts Actor A, A trusts B and C trusts the CA. From those trust statements we can infer that C trusts B.

The FELIX project decided to support two different types of users: the administrators and the experimenters. Each role has a well-defined set of allowed actions and operations. To fulfill the security requirement, the FELIX RO could provide e.g. different interfaces for the different users and roles, or could adopt a sort of role hierarchy which allows some operations (e.g. create new resources) for the higher level, the *administrator*, and others (e.g. reserve available resources) for the lower level, the *experimenter*.

Each entity in the FELIX RO is responsible for authenticating and authorizing requests, thus safeguarding against unauthorized usage. As a third pillar of security, each entity needs to log actions associated with the performing actor. This accounting ensures that all actions can be retraced and malicious users can charged. Also, the data can be used to reenact attacks to learn from it and improve the system.

#### 4.2.1.2 Transit Network Resource Manager

The responsibility of the Transit Network Resource Manager (TN RM) is to support the FELIX architecture with mechanisms to implement network connectivity in particular domains and between them.

In order to deliver the network services in FELIX environment, the TN RM must be integrated with its southbound interfaces with a particular network domain. Such a domain can use different L1/L2 technologies, can be controlled by a Network Management System (NMS) or some specific interfaces or protocols. Since this is up to individual case, it is out of the scope of this deliverable and is considered as purely implementation issue. It is in responsibility of engineers and developers to assure proper integration of TN RM with a domain at the moment of deployment of FELIX system.

The TN RM must communicate with its RO, in order to receive requests and to notify RO about success or failure events. Single TN RM must be in relation to single RO only, while at the same time single RO can have multiple TN RMs under his command. A single TN RM is responsible for particular network resources, which can be called a network domain, and are commonly managed by single entity, i.e. network administrator or NMS.

TN RM usually manages L1/L2 transport networks (rarely L3) which are build of switching devices using frames/packets switching or circuit switching technologies. In particular example of FELIX, they will mostly be Ethernet networks, with assistance of e.g. MPLS protocols. The main difference from SDN RM is that TNRM is not using the reactive paradigm of Software Defined Networking and controls the resources by configuring e.g. VLANs or MPLS paths. TN RM is usually assisted here by local NMS, which may be vendor or domain specific solution.

A single administrative domain managed by an TN RM may be a complex one, which means that it contains not only transit network resources, but also other types of resources. Those resources will be under a control of different, coexisting RMs (e.g. SDN or CR), but still under responsibility of the same RO, as depicted on Figure 4.7.

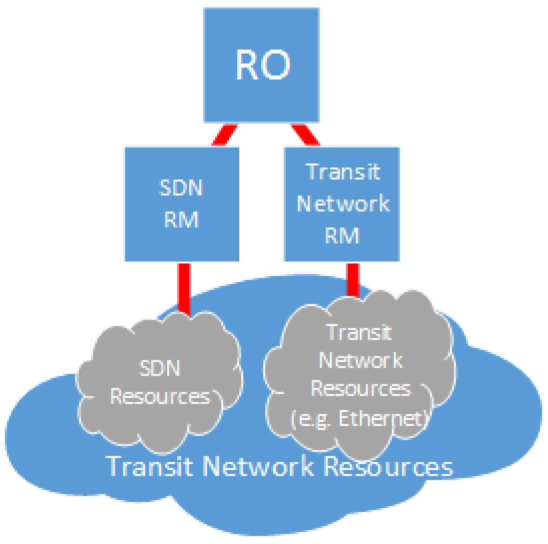
RO
SDN RM
Transit Network RM
SDN Resources
Transit Network Resources (e.g. Ethernet)
Transit Network Resources

Figure 4.7: multiple RMs in single domain

In the example on Figure 4.7, the same domain has both SDN resources, managed by SDN RM, and transit network resources, managed by TN RM. This situation is considered to be a common case in FELIX deployment, and therefore a proper interaction mechanisms between RMs are defined. There are identified several cases of deployment of TN RM component:

1. in purely transit network environment
2. in mixed environment (e.g. coexistence with SDN resources under independent management); in such case two options are possible:
   (a) TN RM sub-domain has no stitching point with other sub-domains -- in such case interaction is not required as there are no common elements of the infrastructure and the sub-domain can't use others' resources

(b) TN RM-sub-domain has a stitching points with other sub-domains -- a slices may require to use resources from multiple sub-domains, which as a consequence requires additional interaction process between RMs (either direct or via RO)

The interaction between RMs may be directly implemented as east/west bound interfaces, or be implemented with intermediating RO using existing interfaces. The second options is preferred for implementation, as it minimizes the amount of implementation efforts required for each RM by eliminating additional specific interfaces. In general the RO should be responsible to delegate specific requests to proper RMs under his management, so that each RM has to perform actions only on resources, which are in this RM control. This way e.g. an TN RM does not need to be aware of SDN resources and vice versa, a SDN RM does not need to know about transit resources in a domain. The management of stitching points between resources types are the key to proper execution of the request and realization of the reservations. In domain with multiple RM, it is also possible that not all RMs are involved in particular reservation, as not all types of resources are used. E.g. if an SDN + transit network domain is used only for transport L2 traffic, the SDN resources may be not used, while TN RM will be asked to create a specific edge-to-edge connection through the domain.

The TN RM must have knowledge and ability to communicate with other TN RMs, controlling the adjacent domains, in order to perform multi domain reservations, as depicted on Figure 4.8.

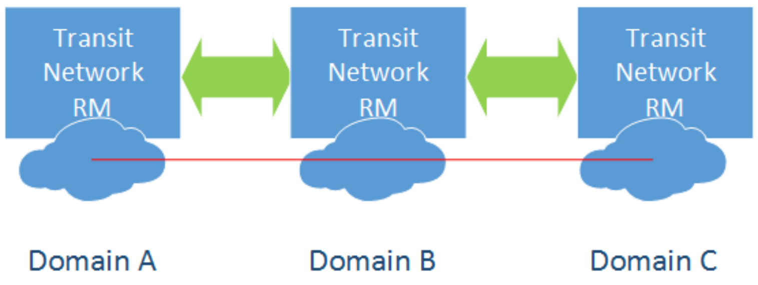

Figure 4.8: inter-domain communication requirement

A connection passing domains A, B, and C requires synchronization of efforts and configuration of resources in three different domains, while each domain is under control of independent NMS, may use different technology and have different policies. In order to implement such circuits all TN RMs must collaborate by exchanging information about their resources availability and configuration technical details (e.g. common VLAN or LSP identifier). From the architecture point of view, it is unimportant whether TN RMs exchange messages directly or via a ROs hierarchical tree, and whether TN RMs can contact any or only directly adjacent TN RM. This particular issues can be decided during the implementation process. Having a direct communication channels between TN RMs in different administrative domains may raise several issues, mostly policy and security related. Also RO responsible for different domains may be interested about actions taken by its particular RMs, which were requested by neighbor domains. Using a ROs as an intermediate point for communications between RMs will give more control over the interaction process, however this approach may increase the time required for communication. FELIX is keen to reuse existing software and tools, which may have specific solutions for this case and the implementation of the system will rely on its mechanisms. E.g. one of the candidates for implementation of TN RM interfaces is OGF NSI Connection Service protocol, which will require TN RMs to behave as BoD services, which can request resources from each other (east/west bound interfaces).

The TN RM must have knowledge on its own resources, as well as an overview of all other transit network domains and its connectivity, in order to enable global inter-domain path finding and prediction of resources usage. Since TN RM will be mostly used in inter-domain context, a proper mechanisms must be implemented to share such knowledge between TN RMs in a dynamic manner. The shared information should be abstracted, so that it not discover domain internals (e.g. private information or key elements related to the security of the network) and give at least an overview of reachability and interconnectivity information. Such information should include abstracted interfaces where connection can be terminated, throughput of particular exposed links, and inter-domain connectivity information. In the simplest cases a single domain can be represented by a cloud (full mesh topology) with abstracted interfaces providing the links to adjacent domains and/or entities to which service can be delivered (usually particular interfaces or services on switches or routers within a domain, from where service can be delivered to particular users, applications or installations). The Figure 4.9 depicts in very general way an example of such abstraction.

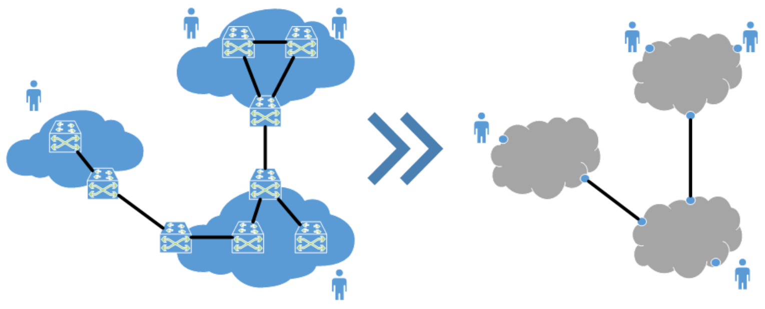

Figure 4.9: General example of topology abstraction

Relaying on abstracted topology view a single TN RM will be able to define a route of a particular inter-domain L1/L2 circuit using path finding functionality. TN RMs are assigned to particular domains, and this information can be stored either in topology data or in separate lookup service. Therefore having known which domains will be involved in implementation of a single reservation, an TN RM will known which other TN RMs should be contacted and how.

The TN RM must be able to collaborate with domains, which are not under FELIX framework management, i.e. network transport domains. Therefore it is even more important for TN RM to implement standardized interfaces for inter-domain reservation. An example of such situation is depicted on Figure 4.10.

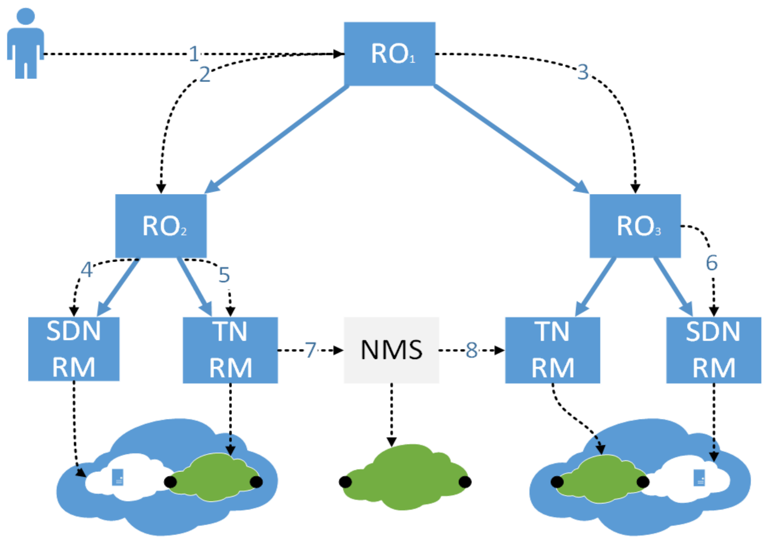

Figure 4.10: Network reservation through non-FELIX domain.

The most left and right domains are under FELIX jurisdiction and includes both SDN RM and TN RM under control of proper ROs. The user request is forwarded to those RMs via ROs hierarchical tree. The middle domain is a network transport domain which is not aware of FELIX framework, has no TN RM and RO assigned to it. In order to implement a production connection between edge domains, the TN RM in both of them must implement some standardized interfaces, which will enable them to communicate with the middle domain NMS. Such a candidate may be for example an OGF NSI CS, previously mentioned. In the particular case depicted on Figure 4.10, the most left TN RM is responsible for path finding, which includes also non-FELIX resources, and then forward the request to middle and right domain. One can notice that $RO_3$ does not request a transit network service from its TN RM. This is caused by the fact that in this example it was assumed that a NSI like service was used, where circuit creation is requested at single point only (at $RO_2$ TN RM) and it is responsible for whole inter-domain circuit creation. The communication is performed directly between TN RMs or adequate instances (NMS of the middle domain here). Otherwise the circuit could not be created as FELIX is not explicitly controlling the middle domain and ROs cannot communicate with it to send a request.

In order to manage the reservations, especially in distributed manner, the TN RM must have an internal state machine for reservations. There are identified tree state machines for reservation (RSM), provision (PSM), and life time (LSM). The state machines is depicted on Figure 4.11, Figure 4.12, and Figure 4.13.

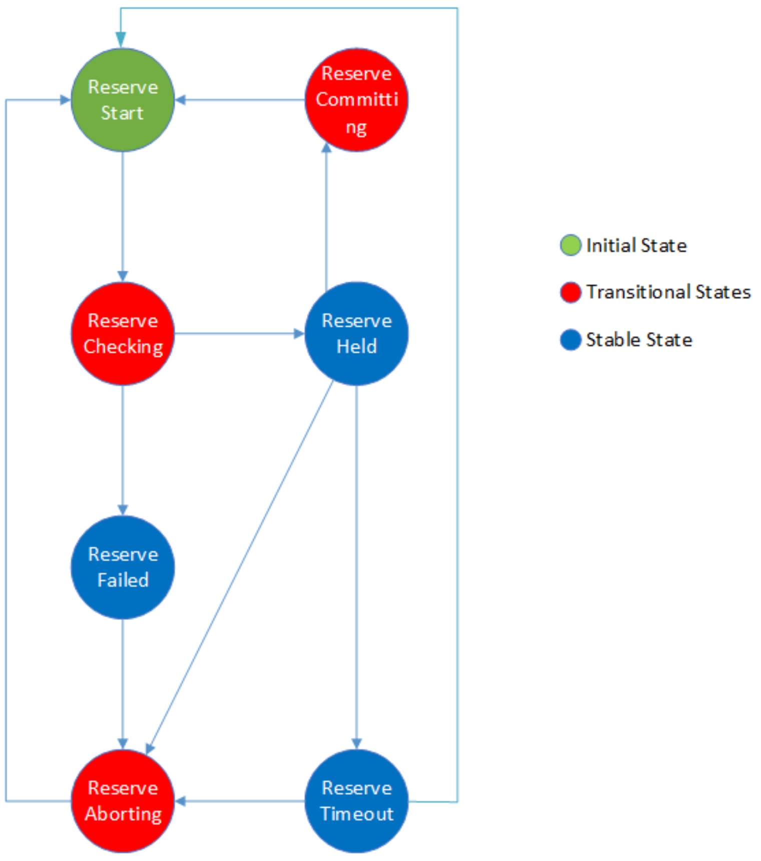

Figure 4.11: TN RM reservation state machine

By the RSM, reservation is made by holding resources between start time and end time of the reservation. The resources held are made available when the PSM is in Provisioned state, and a data plane connection is activated. The resources are made available if and only if the PSM is in the Provisioned state AND the start time < current time < end time.

The PSM is designed to allow resources to be repeatedly provisioned and released in data plane, while being

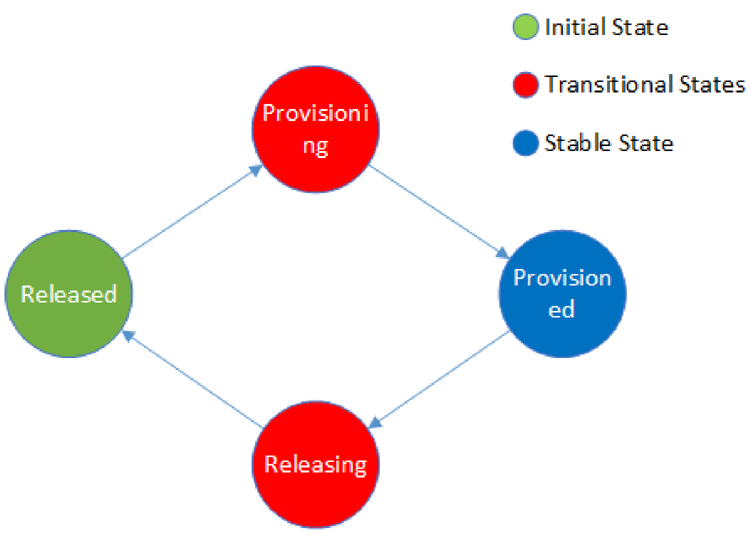

Figure 4.12: TN RM provision state machine

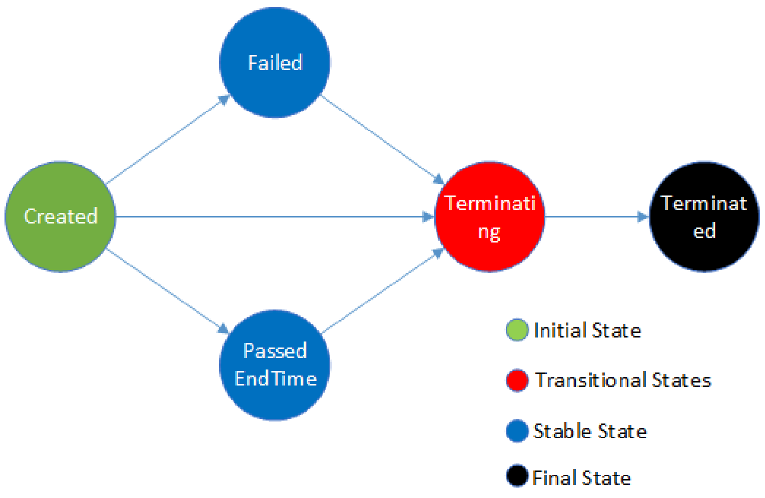

Figure 4.13: TN RM lifecycle state machine

still booked under particular reservation. The PSM transits between the Provisioned and the Released stable states, through intermediate transition states. An instance of the PSM is created when an initial reservation is committed, and at that time it starts in the Released state. The PSM transits states are independent of the state of the RSM. Note that the transition to the Provisioned state is necessary, but on its own is not sufficient to activate the resources (i.e. made available).

The LSM is used for processing events related to terminate the reservation, i.e. on request from external entities (like end users).

The FELIX TN RM should cooperate with supervising RO and peering TN RMs and thus obey the FELIX security aspects related to an Authentication Authorization Infrastructure (AAI). There must be established a trusted relationship between particular TN RM and all other communicating components, the communication must be secured and sender/receiver should be able to be uniquely authenticated. The mechanisms should also support authorisation mechanism, restricting access to some TN RM functionality basing on implemented policy. The

FELIX framework will provide a dedicated AAI infrastructure which enforce the proper security mechanism to be used no only by TN RM by also all framework components. FELIX will use a certificate-based infrastructure and provide a CA (Certificate Authority) entity. TN RM needs to log taken actions and an entity and/or user associated with them. This very basic form of accounting ensures that all actions can be retraced and malicious users can charged. Also, the data can be used to reenact attacks to learn from it and improve the system security.

#### 4.2.1.3 SDN Resource Manager

The **SDN Resources Manager** for the FELIX project will provide it the mechanisms to manage the network architecture inside a domain with SDN-enabled hardware (e.g. OpenFlow switches and routers) it is important to notice that the SDN RM belongs to the **FELIX Space**.

In SDN RM scope, FELIX does not care about the network physical resources. Simplifying, in an SDN domain, like an OpenFlow network, the users (experimenters) can control the network behaviour by actively updating the flow tables of the network elements. This update is usually done by a controller, a software tool that analyses the incoming traffic to a network resource and decides where to send it according to the user's will.

The issues start arising when several users want to use the same resources. It is then not obvious how the related traffic is isolated, so different controllers can only manage only their respective packets. In this architecture, this is achieved by deploying a special purpose controller between the network elements and the users' controllers. This special purpose controller acts as a proxy deriving each user's traffic to its own controller. Each user has his traffic assigned to a flowspace, so it is distinguished from other users' traffic. This flowspace can be a range of source or destination IPs or MAC addresses, TCP or UDP ports, etc.. One way to separate the traffic is assigning a VLAN tag to each packet. In this case, the special purpose controller inspects the incoming packet, identifies the VLAN tag and sends it to the corresponding user's controller.

The main functions of the SDN RM can be described from the network manager's and the experimenter's point of view. For the network manager, the SDN RM will provide the managing functionalities for the network resources. It can define the special purpose controller of the testbed, approve or deny the experimenters' flowspace requests, etc.. For those managing functionalities we see that the SDN RM contains an OpenFlow Resource Manager (OF RM). For the experimenter, the SDN controller offers an interface to define the creation of slices of OpenFlow resources. An OF resource slice is formed by a flowspace that isolates the experiment traffic and a controller that manages that traffic. These functions (isolation and management) are performed by the special purpose controller (e.g. FlowVisor [3]).

Aside from the main functions described above, the SDN RM fits into the the FELIX AAI (Authentication Authorization Infrastructure). This infrastructure provides the necessary mechanisms to authenticate, authorize users, as well as provide accountability. In order to offer these functions, FELIX implements a Clearinghouse, which builds the start of a trust chain. This chain can then be used to verify the identity and privileges of actors. By using a certificate-based approach, FELIX gathers the flexibility to federate the SDN islands easily. By installing Clearinghouse certificates actors can be verified against different Clearinghouses. Please see the "FELIX Resource Orchestrator" chapter for more information.

The next figure depicts the structure. Additionally to the shown entities, the experimenter interfaces with the AAI system. In order to create namespaces for slices and assign users to them (including privileges), the experimenter can use a client (a GUI or CLI) to make the calls at the AAI system. This client also handles the interaction with the SDN RM.

The network manager sets the Special Purpose Controller of the testbed and other administrative/configuration options. This Special Purpose Controller is connected to the OF devices (e.g. switches) and when a packet arrives to a switch and there is no entry in the flow table, it is sent to the Special Purpose Controller which, with its slicing logic, re-sends the packet to the corresponding experiment controller.

***Description of the OF RM (OpenFlow Aggregate Manager)***

The OF RM allows experimenters to allocate OpenFlow resources based on slicing: it is used for administrating OF resources associated to slices. The OF RM is used in order to handle (create, analyze, approve, reject, disable,

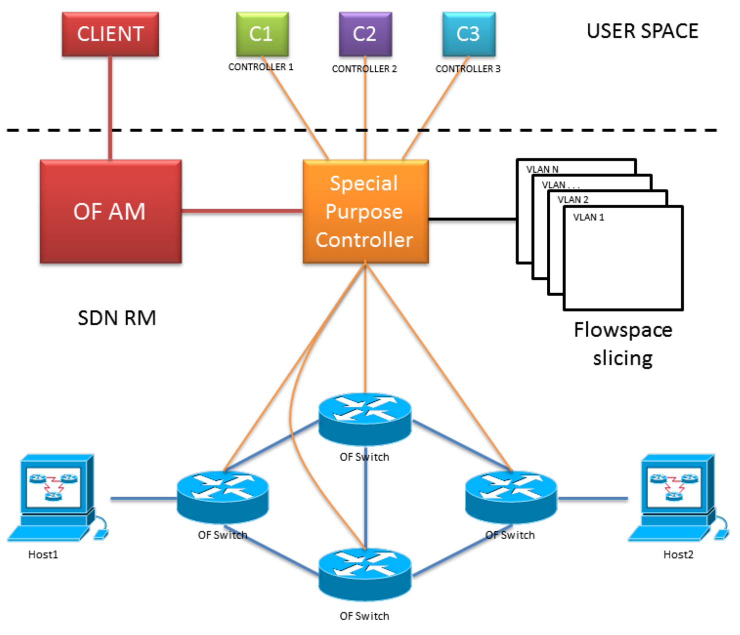

Figure 4.14: SDN Manager Schema

delete, list) OpenFlow switch slivers, i.e. instantiations of OpenFlow slices.

As an AM, it should be easily extensible: interfaces on the northbound (e.g., administrator) and southbound (e.g., FlowVisor) interfaces can be added to cover the needs of the testbed administrators (IMs). Essentially, the OF RM provides support for pluggable administrative and configuration interfaces, used to manage the accounting and resource allocation for experimental slices running on top of physical OpenFlow network substrates. It should be fairly lightweight and can run on the same system as the slicing mechanism (i.e. FlowVisor) without additional hardware requirements. Testbed administrators can also run it on a separate system for isolation purposes, e.g. so that if they need to reboot the OF RM server, that doesn't affect slicing mechanism. The OF RM should support some very useful features regarding interfaces with external components such as URL handlers (namespace/call etc.) and event-handling (slice expiration, event codes, etc.).

As core services, it provides functionalities to setup logging and instantiate northbound APIs and plug-ins, credential checking and verification, southbound API for creating slices, inspecting slicing mechanisms, changing the slice FlowSpace, etc. (essentially, the south-bound interface of OF RM, used to communicate with the slicing mechanism and Virtualization Tool) and a library for slivers, controllers, FlowSpecs, datapaths and miscellaneous objects of the OF RM. It also contains modules for handling FlowSpace allocation, user authorization, DB configuration, exception handling, logging, tracing and JSON encoding and validation.

1. Use AM API to inspect aggregate
2. Determine available datapaths and ports (i.e., available OpenFlow resources)
3. Construct OpenFlow resource request
4. Send RSpec request to AM via AM API, requesting the creation of a sliver with the required resources
5. Wait for admin approval or rejection of the experimenter’s sliver
6. Start experimenting (i.e., utilizing the instantiated sliver)

From the OF RM administrator’s side, the workflow is as follows:

1. Configure main administrative credentials (admin password, etc.)
2. Trust a AAI system for user authorization, configure slice authority credentials
3. Setup the OF RM to talk to slicing mechanism
4. Configure local site tags
5. Annotate datapaths with appropriate information (location, etc.). Optional
6. Setup administrative e-mail notification (e.g., informing about new sliver requests)
7. Set sliver auto-approval policies and configure sliver FlowSpace analysis engine(s)

A full sliver's life-cycle as derived by the OF RM's functionality inspection is the following (this mainly applies to slivers which are not approved automatically, but only after an administrator's intervention):

1. User: create sliver passing the resource description
2. OF RM: parse incoming resource description and return
3. OF RM: send e-mail to admin and user that the resource has been created and its approval is pending
4. Admin: approve-resource, notify the associated parties (admin, user)
5. Deletion of sliver is carried out by any of these means:
   (a) User deletes resource
   (b) Admin deletes resource
   (c) OF RM: resource expires automatically in case none of the previous steps were taken

Pending, active and even deleted resources can be inspected. Related resource info that can be shown via the OF RM includes basic resource details, resource descriptions (i.e RSpecs), FlowSpecs and FlowSpaces.

Resources can be shown (listed), approved, disabled, rejected or deleted via the OF RM.

#### 4.2.1.4 Computing Resource Manager

The function of the **Computing Resource Manager** is to provide a method to assign, setup and configure computing resources inside a FELIX island. It manages physical computing resources, and also the configuration of slicing mechanisms (e.g. common hypervisors or other virtualization stacks) and computer resources as seen in User Space (OS images, network interface configuration, etc.)

The Figure 4.15 shows C RM operations for three typical scenarios.

- Single hypervisor (left), the C RM instructs the hypervisor to create (or remove, migrate, …) computing resources (VMs) when needed. Each computing resource belongs to a specific slice.
- HaaS scenario (middle), the FELIX infrastructure provides hardware-as-a-service. The computing resources managed on HaaS level are not VMs but entire IaaS stacks. Contrary to the figure, there may also be scenarios where the user completely configures VMs from within the experiment, bypassing the C RM. Similar for OS images which may be user-provided in this case.
- Physical machine (right), an entire physical machine is assigned to a slice.

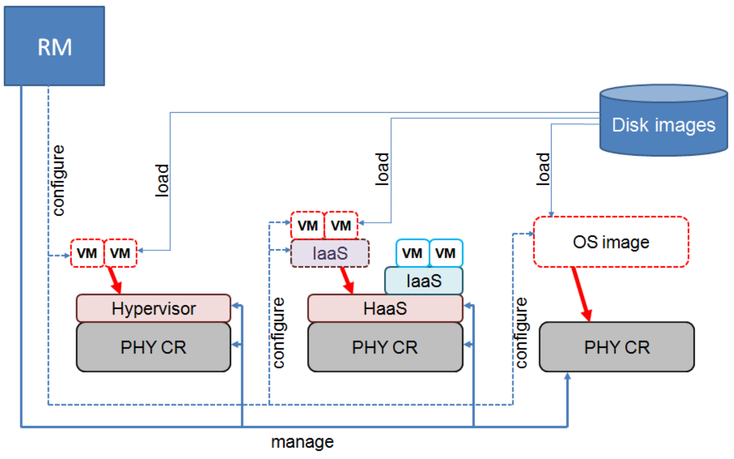

Figure 4.15: Computing Resource manager operations

In all scenarios, a disk image repository is available, which is used to distribute OS images to computing resources (e.g., already containing some typical FELIX user tools, slice resource controllers, etc.)

Management of physical computing resources also provides methods for rebooting machines, remote control (of a machine's console), or hard power on/off of a machine experiencing problems, for example using a networked PDU (power distribution unit). Management is typically only performed during problems, or when a slice is created, destroyed or modified.

***Configuration of computing resources***

Whereas most of the management matches slice life cycles, the functionality providing configuration and loading disk images is active anytime a slice computing resource is created or modified. Migration of computing resources to other islands may also require reconfiguration.

Configuration includes:

- setting of unique information in the computing resources, such as IDs, SSH keys, IP and MAC addresses
- setting up monitoring of computing resources
- configuring of network interfaces of the computing resource, and setting the underlying resources (e.g. hypervisor, HaaS platform, physical machine...), such that those interfaces are bridged onto the physical interfaces that are actually connected to SDN zones in the FELIX island
- if necessary, configuring of a slicing mechanism in this bridging, in case multiple computing resources or slices have to share a single physical interface, typically using a (software-based) SDN solution inside the virtualization platform. Once the SDN solution has been properly set up, it becomes an SDN resource which is managed by the SDN Manager.

Configuration may be done by writing to a computing resource's disk images before staring it, or by having scripts in the computing resource which retrieves configuration data from the C RM. The loading of disk images itself can be managed by the C RM, or it can be left to the computing resource itself, by providing it with a small stub OS which retrieves the appropriate disk image from the repository at first boot-up.

### 4.2.1.5 Monitoring

The purpose of the monitoring framework is to retrieve, aggregate and store the monitoring data of the slices containing resources from multiple testbeds. The type of resources to be monitored are computing resources, SDN resources and TN connectivity between different FELIX islands or facilities.

We have deeply analysed different kinds of monitoring framework as proposed by other FIRE projects (see "Related work and testbed analysis" and "Appendix A" for reference). According to Fed4FIRE project, we can define two different types of monitoring: facility monitoring and infrastructure monitoring.

*Facility monitoring* includes basic status information about the facility, such as whether server are up and network connectivity to other testbed (or the Internet) is available. This information is generally available in testbed and can be published onto the FELIX GUI (or user portal) for the users and/or testbed managers. Also included in facility monitoring is status information about the functional components of the control and management framework. For FELIX, this means that monitoring of the Resource Orchestrators and Resource managers (and controllers if not already monitored by general facility monitoring) should be implemented and integrated into the monitoring framework.

*Infrastructure monitoring* then concerns the actual resources which are available or provisioned in the FELIX infrastructure:

- Computing resources: available (virtualization) servers, memory usage, CPU load, etc.
- SDN resources: available switches, ports, flowspaces (VLAN, MAC addresses ranges, etc.), usage information (from device counters)
- Transit network resources: available connectivity, endpoints (STPs), bandwidth

The figure below shows the general architecture for the (infrastructure) monitoring framework. The framework consists of monitoring agents (M) and aggregation infrastructure (DB, database).

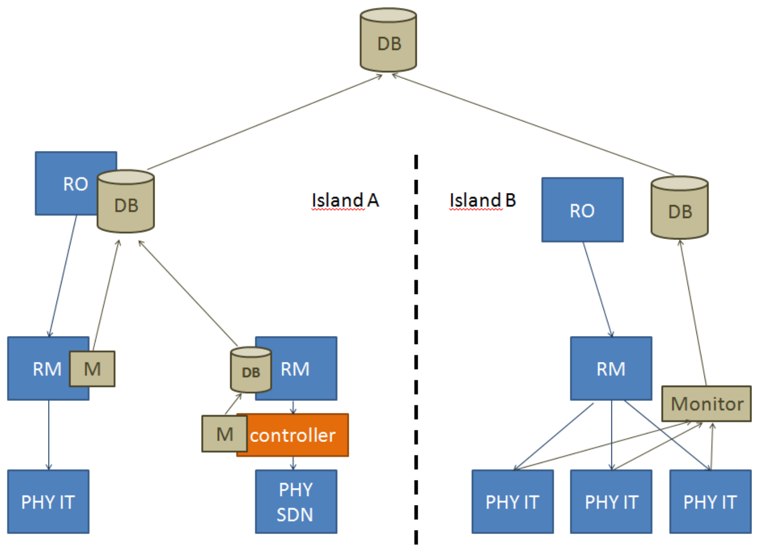

Figure 4.16: Monitoring architecture

***Aggregation***

The monitoring of resources creates extra state information inside of the testbeds and FELIX federated infrastructure. For some types of resources this monitoring data can match the provisioning information present in the Resource Managers. For example, VLAN IDs that are requested through the FELIX architecture (using SFA) should match utilized VLAN IDs as monitored from SDN resources. However, for several reasons there can be a discrepancy and there is indeed merit in creating additional state through monitoring. For example, there may a problem with equipment, configuration or the provisioning mechanism that causes inconsistencies between provisioned (by the RM) and available resources (as per the monitoring framework).

Moreover, there is the possibility that CR, SDN and especially TN infrastructure is shared with additional (federation) architectures for support of experiment outside of FELIX scope; similarly, some resources may be administratively reserved by testbed operators. In this case monitoring is required to find out the actual availability of resources.

Lastly, some monitoring information depends on the real usage of experiments and users. For example, for computing resources, the RM will now about provisioned VMs, memory etc., but the actual memory load will depend on the experiments run by the users (and whether VMs are active for the entire duration of the experiment). For TN resources, connectivity is provisioned with a certain bandwidth, but the actual bandwidth usage will be variable.

As the figure shows, this additional state is kept in databases (DB). To minimize replication of state, aggregating information into additional higher-level databases should be avoided. However this may be inevitable because:

- aggregating information reduces load on the monitoring framework
- since physical infrastructure may be shared with other testbed control frameworks (now or in the future), the FELIX monitoring framework may not have direct access to monitoring agents and have to rely on forwarded or aggregated monitoring information (originating from the other control framework)
- aggregating monitoring information provides a single source for FELIX users to retrieve (current or historical) resource availability.

Where possible, aggregating databases may act as caches only, meaning they can be reconstructed from the lowest level databases closest to the Resource Managers and monitoring agents. The Fed4FIRE project currently proposes OML for the aggregation of monitoring information.

***Monitoring agents***

On the figure three types of monitoring agents are shown:

- agents integrated into the Resource Manager of the corresponding resource (ideal case for FELIX framework)
- agents integrated into a controller of the corresponding resource
- agents not coupled to RM or controller, but implemented as a stand-alone services (nevertheless reporting to FELIX databases or RMs).

The monitoring agent may need to be integrated into the controller if the resource requires this. For example, for SDN OpenFlow resources, monitoring information (e.g. flow counters) are exchange through the OpenFlow protocol, and an OpenFlow switch connects to only one controller, so it is not possible to bypass the SDN controller and implement the monitoring agent into the RM.

In some cases (e.g. computing/storage infrastructure), testbed operators may use centralized monitoring appliances (e.g. Zabbix, ZenOSS) that cannot be easily integrated with FELIX components because of technical issues or because of being shared with other control frameworks.

In the cases the monitoring agent could not be directly integrated with the Resource Manager, aggregation should allow integrating the higher-level databases to be integrated/managed in/by a higher level Resource Manager or Orchestrator. Alternatively, a monitoring agent may report directly to a corresponding Resource Manager so that there is no state outside of a RM.

***Slice Based Federation integration***

The availability of resources according to the monitoring framework will be exposed through the FELIX Management and Orchestration Architecture. Both higher-level Resource Orchestrators and users (and gui) can use this functionality to decide on resource allocation. Aside from implementation of this functionality, appropriate resource specification languages should be defined for this integration.

#### 4.2.1.6 User Access/GUI

The FELIX User Interface is in charge of both exposing and allowing access/management to the different testbeds resources, such as:

- Computing/storage resources: servers, VMs (hard disk, memory), storage units, etc.
- SDN resources: switches, ports, flowspaces (VLAN, MAC ranges…), etc.
- Transit Network resources: connectivity, endpoints (STPs), bandwidth, etc.

An UI offers a friendly way to control the lifecycle of an experiment for different type of users, namely experimenters and administrators. In the experimenters' side, they are given the necessary permissions to list the resources, easily select a subset of those in order to allocate or provisioning them, use and finally freeing those. Addressing the needs for the administrator, it would be possible to perform tasks such as configuring resources and policies, activating/deactivating those, monitoring the resources in order to take further action, approve resources requests from the experimenters, etc.

As for the different expressions that the UI shall take, this is to be analyzed in order to make available a subset of those. Each expression provides different benefits, for example a Graphical UI offered through a web portal can offer a higher user experience while the command line tool benefits are the automation, as well as the possibility of allowing mobile access to extend the project scope.

While the User Interface allows access to resources it also communicates directly with the Authentication and Authorization Infrastructure (AAI) module in order to control who can access those. That is, the UI is closely related to the AAI system, which, in a Slice-based architecture stores the following relationships in a registry (user:permission, permission:resource) and implements the necessary logic in the AAI module to grant or reject access to a user given its credentials and the resources identifiers to be accessed.

Then, the aforementioned AAI module is the ultimate responsible of granting access to the resources, but it can be further extended by policies, which are a set of rules defined by the administrators to implement an upper-level control on the resource usage (e.g. defining a maximum virtual memory value for a VM resource or a maximum number of flowspaces).

Integrating the UI with a policies tool is advisable, as it also might be desirable to allow access to some AAI information through the GUI.

The prototype for the User Interface would be reflected in the deliverable D3.4 ("End User Tools and API").

### 4.2.2 Slice Resources Controller

The Slice Resource Controller in the context of the FELIX architecture includes all functionality, APIs and applications that allow the experiment user to control the slice

- from within the experiment (User Space),
- needing support from the FELIX framework, provided by some component(s) in the FELIX Space.

For example, consider an experiment that creates dynamic traffic streams across several SDN islands and transit domains during the runtime of the experiment. The experimenter may set up some traffic generators in User Space, possibly use a custom controller to configure bandwidth patterns. However, this is contained within user space, very specific to the experiment and therefore not within scope of a Slice Controller.

On the other hand, once this bandwidth is generated, the user may want to request some dynamic bandwidth across transit domains, or define some SDN routing scheme. For this, a custom (user-provided) controller can be used, or a generic FELIX provided controller may suffice, but in both cases these actions at some point do require access to the FELIX infrastructure outside User Space, therefore needing a Slice Controller (or Transit Network and SDN controllers respectively) which takes the User Space bandwidth requests and passes them on to the FELIX Space (e.g. NSI and OpenFlow components).

Put more clearly, there are a number of operations that are considered 'control' of some sort from the user perspective:

- Management of resources, using the RO and related components in FELIX Space
- Slice control; that is, control of the slice, namely, dynamically adding or removing resources.
- 'other' control, for FELIX use cases this is Network Control (e.g., OpenFlow) and Computing Resource Control (control of VMs, physical nodes, etc.)
- control implemented through user tools

Out of scope as far as the architectural discussion is concerned, are user tools as well as Network and CR Control. User tools are implemented by the experimenter; Network and CR Control use existing solutions such as OpenFlow. FELIX Space however does have provisions (were necessary) to configure the slicing mechanisms properly for these types of control to function. In view of the use cases, FELIX may create or propose some standard implementations for these type of control functions for the respective use cases, however these implementation build onto the FELIX architecture but are not part of it.

The scope of the Slice Resource Controller is the slice control. Control over the slice offers the ability to the user to use well-defined APIs to add/remove resources (e.g. for virtual machines), request additional access to the network (e.g. flowspaces), change connectivity or bandwidth reservations (e.g. over the transit network), etc. It may also be used to reconfigure part of the slice, without a changed in reserved resource: for example, a tool such as VeRTIGO can provide a means to offer logical topology changes to the user slice; another example is 'virtually' cutting/repairing network connectivity to simulate network failure scenarios (such as the disaster recovery use case).

***Types of Slice Controllers***

Different functionalities offered by FELIX to the slice are exposed through different types of slice controllers.

- CR controller: for virtualized computing resources, the hypervisor and its API toward the guest kernel (inside the slice) provides fairly transparent access to computing resources outside the slice (e.g. disk access, CPU, physical network ports). The hypervisor (i.e., the controller) will support nested virtualization for use case that requires this. For testbeds providing physical computing resources, there may be dedicated control; for example, the Emulab virtual wall at iMinds provides node control from the tevc (Testbed Event Client) to control and signal nodes.

- Transit Network controller: allows setting up (or tearing down) dynamic bandwidth using the transit network infrastructure outside the slice. This controller may be implemented mostly as functionality offered by the FELIX control framework. In any case, an important aspect is also the connection and stitching of transit network and SDN zones, testbeds and/or domains (e.g., VLAN translation).

- SDN controller: allows controlling a slice of SDN resources (flowspace), and use SDN techniques to route and/or switch slice traffic over the physical FELIX infrastructure, over SDN-enabled switches and to/from transit networks. The SDN controller consists of the controller provided by the experimenter, and the part of the FELIX SDN Resources Manager which interfaces with the experimenter's controller, i.e., provide a known protocol such as OpenFlow, and adapt the slice SDN actions to the FELIX SDN resources to support slicing, for example flowspace filtering, translation, logical topology support etc.

***Types of control APIs***

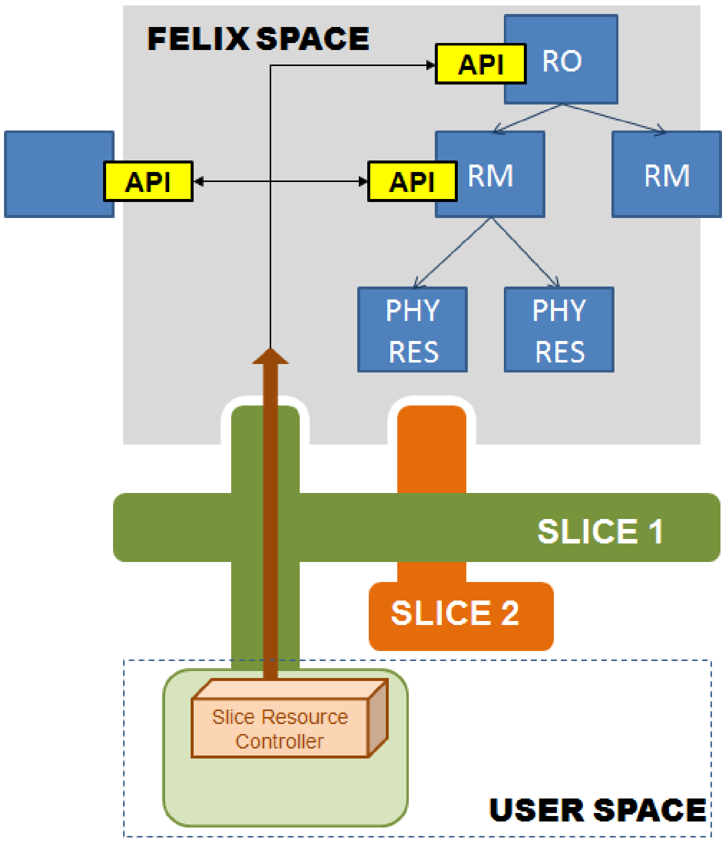

Figure 4.17: Slice control and APIs in the FELIX architecture

In most cases, the slice controller will be implemented as two entities: one inside the User Space, and one residing entirely in the FELIX SPACE, outside of direct control of the user. Between the two entities, there is a control API.

As shown on Figure 4.17, the management architecture in FELIX Space is used to provide the slice for the user experiment. The Slice Resource Controller in run inside User Space, which is supported by the resources of the slice. API end-points are visible from the slice (and User Space). As there is conceptually overlap between control and management function (for example, both control and management can be used to add resources to the slice), in many case the control functions in FELIX Space will be part of respective Resource Managers or Orchestrators. In these cases the APIs are attached to these RO/RM. In some cases the type of resource control is may not be a function of an RO/RM, and a separate controller needs to be provided in FELIX Space. In other cases still, some legacy control may be involved which falls outside of FELIX Space. Then the API can either talk directly to components outside FELIX Space, or the communication can be proxied through an RM or other FELIX component.

***Mapping of Slice Controllers and Resources (Resource Controllers)***

Through the use of APIs, slice controllers are mapped to FELIX infrastructure resources and resource controllers. This mapping can be one-to-one, one-to-many and many-to-one on different abstraction levels. An example is shown on Figure 4.18, separating User Space from the rest of FELIX architecture via APIs, for two slices over two SDN islands connected through a transit domain.

Because of the slicing mechanism, multiple slices are supported on top of a single computing resource (VM server), SDN resource (switch) or NSI resource (links and switches). Therefore at the very least, many slice con-

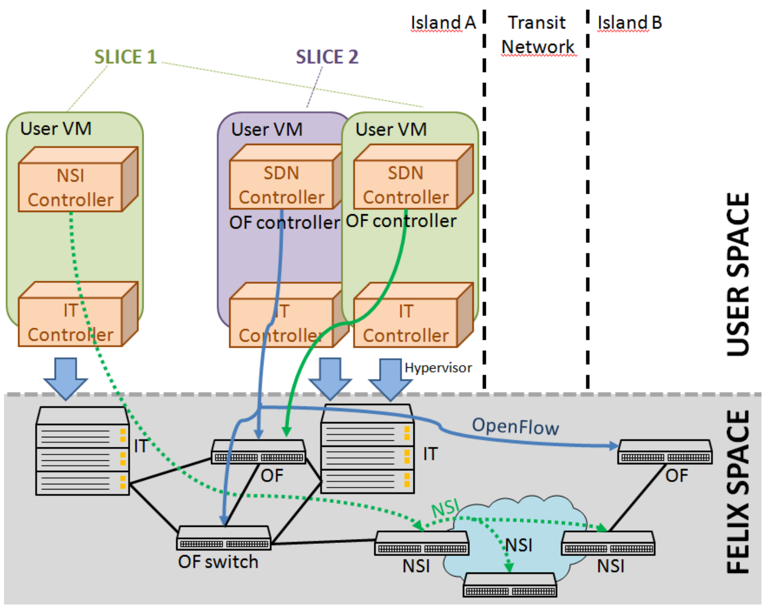

Figure 4.18: Examples of mapping between slice controller and resources (resource controllers)

trollers are mapped to the same resource controller in the FELIX control framework; for example, multiple VMs on a single VM server, multiple OpenFlow controllers on a single FlowVisor or other OpenFlow slicing device.

From the viewpoint of a single slice, there can be one-to-one mapping, for example, because of slice isolation, VMs, flowspaces, etc. from other slices are not visible, so a VM's kernel can be fixed to one hypervisor of a single VM server.

Within a slice, a controller can control multiple resources. An experimenter's SDN controller can control multiple switches, possibly even in multiple SDN zones or islands. A Transit Network Controller will -in the FELIX use cases- request connectivity in multiple NSI domains. Using the CR controller, a VM may be migrated to a different VM server (different hypervisor), possibly to VM server in a different island.

# 5 Conclusions and Summary

The FELIX project creates a FELIX Federated Framework which allows users to request, monitor and manage a slice in a distributed, heterogeneous, multi-domain environment.

The first step to achieve this aim is define system requirements for such specific environment which allows to create a virtual infrastructure or a distributed multi-domain slice. Project defined a 5 crucial issues to provide such distributed slice:

1. AAA -- Applying proper security and control mechanism for users in federated environment;
2. Resource management – Coordination of various resources provided by multiple domain heterogeneous resource management systems is required;
3. Resource allocation planning – It is important to create a suitable resource allocation plan of both computing resources and network resources, which can reflect reservation options for user and resource administrator issues, such as cost, energy consumption and load balancing, into consideration;
4. Provisioning – It is important to provide applications with a virtual flat environment, just like a dedicated cluster, using dynamic resource information, such as IP addresses;
5. Monitoring – It is difficult for each user to monitor the usage of distributed and heterogeneous "virtual" resources managed by multiple domains.

The FELIX project has analysed how these issues are resolved in existing project. FELIX has concentrated on 4 European Projects (OFELIA, FIBRE, Fed4FIRE, BonFIRE) and 2 Japanese Projects (GridARS, RISE) focused on the following architectural components:

- General control frameworks;
- Resource discovery, reservation and provisioning mechanisms;
- Experiment managers;
- Identity management tools;
- User interface tools.

The analyse involves also the evaluation of all those aspects in the context of FELIX project. The general conclusion was that FELIX project objective is to aim at large scale federations with merging different resources types at the same time. In comparison, the analysed infrastructures were limited in range or technology, disregarding e.g. long distant network connectivity and dynamic provisioning, where FELIX found his place giving an opportunity to merge and enhance the offered services.

After analyzing existing architectures proposed by the previously mentioned projects FELIX created its own architecture, which is the combination of two spaces (refer to Figure 5.1 and Figure 5.2):

- the FELIX Space (Figure 5.1) – responsible for providing the resources for creating a user slice. It relies on Resource Orchestrators (ROs), Resource Managers (RMs), and the physical infrastructure (test-bed).

The Resource Orchestrators (ROs) is one of the architectural components that is responsible for the orchestration of end-to-end, multi-domain service in the FELIX infrastructure. The Resource Managers (RMs) is the next architectural components that is responsible for charge of controlling a specific type of resources. The architecture defines several types of RMs: Computing Resource Manager, Transit Network Resource Manager and SDN Resource Manager. The next architectural component – the Monitoring Framework – is responsible for collecting

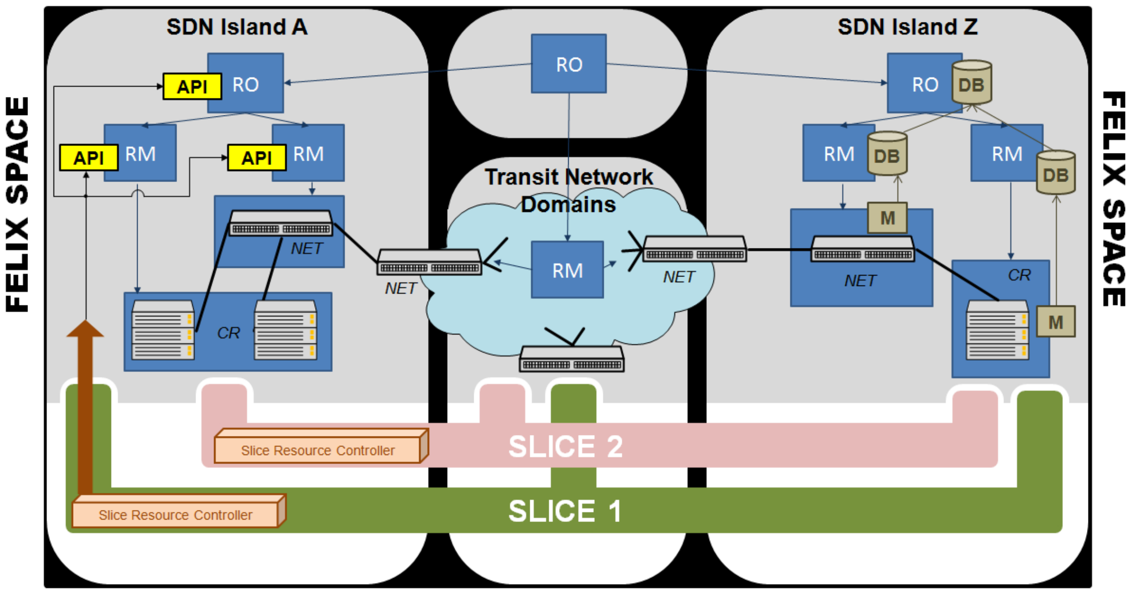

Figure 5.1: key concepts of FELIX Space and components

and managing monitoring data from slices and experiments using these slices. In the ideal case, the monitoring agents and monitoring databases are closely coupled with the RO/RM hierarchy, as shown on the right of the figure (for island Z). The last architectural component – the Slice Controller function – is responsible for managing the creation, modification and deletion of slices. It is implemented as a Slice Resource Controller in User Space, and a set of functions of the RO/RMs. Slice Control is initiated from outside FELIX Space, and a set of APIs is provided to allow this (left part, island A).

- the User Space (Figure 5.2) – consists of any tools and applications which allow to control a slice or execute particular operation on slice by user.

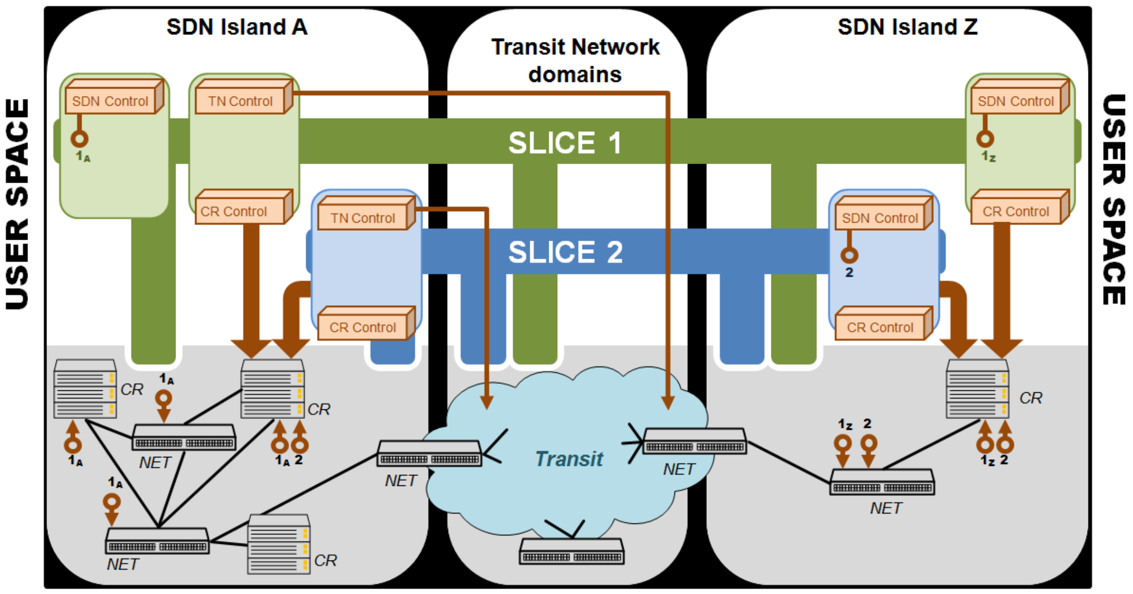

Figure 5.2: key concepts of User Space and components

User Space is supported by the set of virtual resources of the corresponding slice. From the perspective of user Space, the slice consists of a number of IT, transit and SDN resources. These are controlled using Slice Resource Controllers that communicate with FELIX Space through the control APIs described above. The architecture of control inside User Space depends on the specific use case. Slice Resource Control may be spread out over multiple components, and similar types of resources may be controlled from more than one controller (e.g. one SDN resource controller per island, as is the case for slice 1 in the figure).

The FELIX architecture proposed in this document is generic, in the sense it left more specific decisions to be taken during an implementation process. It defines however the main components, its responsibility and dependency, which enables readers to understand the concept of how FELIX will operate. It is crucial to understand that the proposed solution is scalable, and this can be understand in two contexts. It is scalable, in the sense that a built infrastructure can be easily extended with new sites, by simply deploying new ROs and RMs under existing hierarchy. It is also scalable in the sense that the management layer and its RMs are not limited to handle only specific types of technologies or protocols. Despite FELIX proposed three main RM type (CR, TN, and SDN), nothing prevents developers to add new functional block to that, while the interactions and dependencies is already there. In this way FELIX concept can be extended to cover new emerging and future trends in IT, like e.g. Network Function Virtualization [7].

The architecture puts an emphasis on dynamic provisioning and automated components collaborations, which enforces RMs to be autonomic in decision making process, yet enabled for communication and operation in wider, more complex environment. The aspect of federation is extremely important for FELIX, as it will integrate FI experimental facilities on different continents and provide solutions to real existing problems, as defined in Use Cases of D2.1 deliverable [20]. Therefore the automation, independence, and security of all FELIX entities are treated with care and considered in details. The adoption of FELIX architecture at the top of existing FI experimental facility should not be considered as rebuilding the whole existing management layer, but instead as an adding new functionality to enhance offered services, and to expand its range and features.

The proposed FELIX architecture is an input for further implementation efforts, which will prototype and validate specific implementation of the FELIX framework. As mentioned before, the FELIX architecture definition is generic, thus the result of the project will be just one way of realising it in practice. It is envisaged that FELIX project will use NSI CS standard for performing TNRM functionality and SFA for CR management, which are implementation decisions. This document and its statements are the final conclusion of the design and architecture definition efforts of FELIX, however the document and its content may be updated as new ideas or issues will arise during implementation phase of the project.

## References

[1] Bonfire project, http://www.bonfire-project.eu/.

[2] Fibre project, http://www.fibre-ict.eu/.

[3] FlowVisor, https://github.com/OPENNETWORKINGLAB/flowvisor/wiki.

[4] GENI API, http://groups.geni.net/geni/wiki/GeniApi.

[5] GENI, https://github.com/fp7-ofelia/AMsoil/wiki/GENI.

[6] ITIL, http://www.itil-officialsite.com/.

[7] Network function virtualization, http://www.etsi.org/technologies-clusters/technologies/nfv.

[8] NSI Connection Service v2.0, http://redmine.ogf.org/dmsf_files/12970?download=.

[9] RSpecs, http://groups.geni.net/geni/wiki/GENIExperimenter/RSpecs.

[10] SFA2, http://groups.geni.net/geni/attachment/wiki/SliceFedArch/SFA2.0.pdf?format=raw.

[11] www.opennetworking.org/sdn-resources/sdn-definition.

[12] Fibre deliverable, d4.3: Report on the contributions to the federation framework, http://www.fibre-ict.eu/images/stories/deliverables/d4.3%20%20report%20on%20the%20contributions%20to%20the%20federation%2 2013.

[13] Fibre deliverable, d5.1: Report on the detailed design and development of technology pilots, http://www.fibre-ict.eu/images/stories/deliverables/2013_05_24_fibre-d5%20.1%20report%20on%20detailed%20design%20and%20development%20of%20technology%20pilots.pdf, 2013.

[14] S. Bradner. Key words for use in rfcs to indicate requirement levels. IETF RFC 2119, 1997.

[15] Chowdhury, NM Mosharaf Kabir, and Raouf Boutaba. Network virtualization: state of the art and research challenges. *Communications Magazine, IEEE 47.7 (2009): 20-26*, 2009.

[16] A. Farrel, J.-P. Vasseur, and J. Ash. A path computation element (pce)-based architecture. IETF RFC 4655, 2006.

[17] E. Haleplidis and et al. SDN Layers and Architecture Terminology. Internet Draft IETF, December 2013. draft-haleplidis-sdnrg-layer-terminology (work in progress).

[18] Konstantinos Kavoussanakis, Alastair Hume, Josep Martrat, Carmelo Ragusa, Michael Gienger, Konrad Campowsky, Gregory Van Seghbroeck, Constantino Vázquez, Celia Velayos, Frédéric Gittler, Philip Inglesant, Giuseppe Carella, Vegard Engen, Michal Giertych, Giada Landi, and David Margery. Bonfire: the clouds and services testbed. *5th IEEE International Conference on Cloud Computing Technology and Science, Cloudcom*, 2013.

[19] D.G. Perez, J.A.L. del Castilla, Y. Al-Hazmi, J. Martrat, K. Kavoussanakis, A.C. Hume, C.V. Lopez, G. Landi, T. Wauters, M. Gienger, and D. Margery. Cloud and network facilities federation in bonfire. In *The first international FedICI'2013 workshop: Federative and interoperable cloud infrastructures. Euro-Par, Aachen, Germany*, 2013.

[20] R.Krzywania, W.Bogacki, B.Belter, K.Pentikousis, T. Rothe, G.Carrozzo, N.Ciulli, C.Bermudo, T.Kudoh, A.Takefusa, J.Tanaka, and B.Puype. FELIX Deliverableand D2.1: Experiment Use Cases and Requirements.

[21] Atsuko Takefusa, Hidemoto Nakada, Tomohiro Kudoh, Yoshio Tanaka, and Satoshi Sekiguchi. Gridars: An advance reservation-based grid co-allocation framework for distributed computing and network resources. In *Job Scheduling Strategies for Parallel Processing*, volume 4942/2008: 152-168, April 2008.

[22] T.Kudohand, G.Robertsand, and I.Monga. Network Services Interface: An Interface for Requesting Dynamic Inter-datacenter Networks. *OFC2013*, March 2013.

# Appendix A

In this appendix, we discuss further details for each testbed outlined in the previous chapter: "Related Work and Testbed Analysis". In particular, we give a brief overview of each testbed and provide additional implementation details previously omitted for brevity. These details focus on the infrastructure and platform developed in the course of each project, and are described below:

## OFELIA

OFELIA (OpenFlow in Europe -- Linking Infrastructure and Applications) is a pan-european testbed, consisting of the following facilities:

- Berlin, Germany (TUB) – partial replacement of existing campus network with OF-switches
- Ghent, Belgium (iMinds) – central hub, large-scale emulation
- Zurich, Switzerland (ETH) – L2 (NEC) switches mesh, connection to OneLab and GENI
- Barcelona, Spain (i2CAT) – L2 (NEC) switches and optical equipment (ROADM ring)
- Bristol, UK (UNIVBRIS) – national hub for UK optical community; optical (ADVA, Calient), L2 (NEC, Extreme) switches, FPGA testbed
- Catania, Italy (CNIT) – based on NetFPGA and OpenSwitch technologies, with focus on ICN (Infomation Centric Networking)
- Rome, Italy (CNIT) – based on NetFPGA and OpenSwitch technologies, with focus on ICN -- under deployment
- Trento, Italy (CREATE-NET) – a city-wide distributed island based on L2 (NEC) switches and NetFPGA; opt-in users via heterogeneous access technologies
- Pisa, Italy (CNIT, 2 locations) -- based on NetFPGA and OpenSwitch technologies, with focus on Cloud Data Center management -- under deployment
- Uberlândia, Brazil (UFU) -- under deployment

Based on OpenFlow 1.0, OFELIA offers a private test environment for the development and testing of new network applications using novel topologies. It integrates OpenFlow-enabled hardware devices of various vendors (e.g. NEC, ADVA) and sets of virtual machines for traffic generation.

OFELIA's facilities are interconnected to a single network distributed across Europe. The connectivity between the islands is based on Gbit/s Ethernet tunnels.

### OFELIA Control Framework

The OFELIA Control Framework (OCF) is a set of software tools for testbed management. It controls the experimentation life cycle; including reservation, instantiation, removal, configuration and monitoring.

The control framework hides the complexities involved in single and federated island setups, yet still providing enough information so that experimenters can program their environment using heterogeneous, scalable resources. It enables allocating resources and running experiments in the entire OFELIA facility.

OCF features the full software stack: front-end, clearinghouse and resource managers (AMs). It also provides support for management of OpenFlow, Virtual Machine (currently Xen-based) and Emulab resources.

The testbed control framework front-end is a tool called **Expedient**. The island managers will use it to configure and assign resources to projects as well as manage user data and credentials; while the experimenters will use it to configure, start and stop their experimental slices as well as updating their own user, project and slice data.

Expedient communicates with every **Aggregate Manager** in order to perform the resource provisioning. OCF provides a base class for the AMs: AMsoil, a pluggable system that determines a structure and provides the necessary modules for handling incoming communication and easing the resource management.

The core idea behind OFELIA Control framework (OCF) architecture is to provide a federated experimental facility capable of provisioning isolated virtual experimental infrastructure on OFELIA campus islands. OCF is characterised by its modularity ands abstracted implementation. These features enable an incremental development whilst retaining consistency.

It fulfils important functional goals, such as:

- Maintaing autonomy of islands
- Unique experiments identified by independent, isolated slices
- Individual island policy management
- Opt-in resources for experiments
- Federation between other projects (e.g. Import & export resources from other facilities)
- Instantiation of a generalized virtual topology completely decoupled from the physical islands, based on switches, links and virtual machines

## OFELIA Architecture

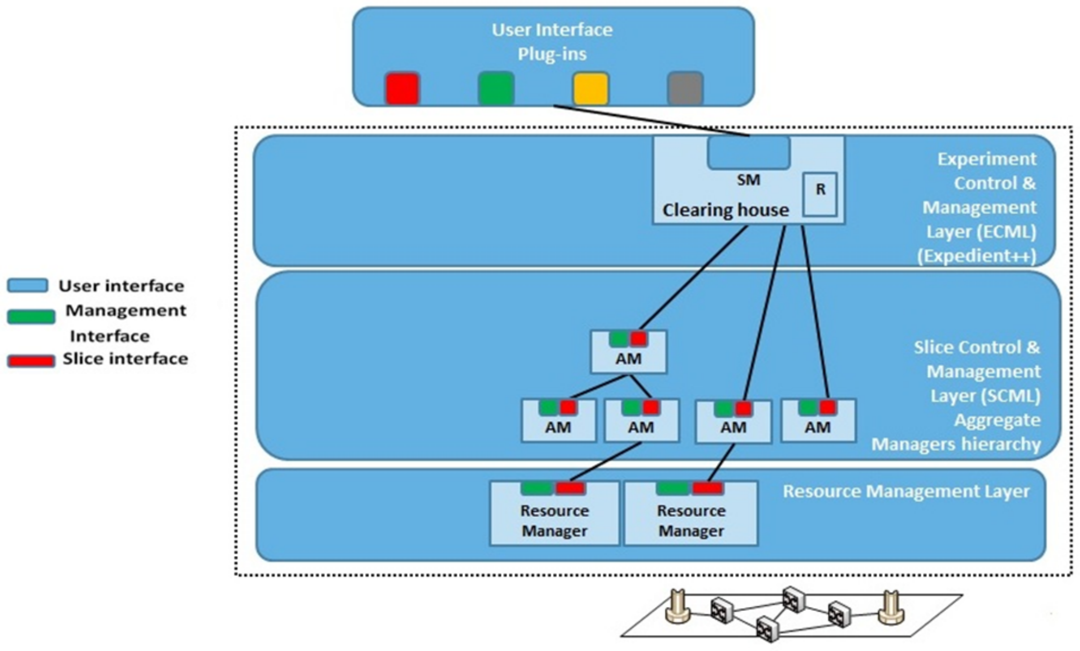
User Interface
Plug-ins
SM
R
Clearing house
Experiment Control & Management Layer (ECML) (Expedient++)
AM
AM
AM
AM
AM
Slice Control & Management Layer (SCML) Aggregate Managers hierarchy
Resource Manager
Resource Manager
Resource Management Layer
User interface
Management Interface
Slice interface

Figure A.1: Individual Island Architecture

Any individual island within the OFELIA facility will use the same control and management architecture to provide experimental services. The experimental, control and management architecture of a single OFELIA island can be divided three layers. The three layers (starting from the top) are:

- **Experiment Control and Management layer (ECML)**: the uppermost layer beneath the user interface. The component at this layer is the clearing house (slice manager and repository), which will contain information about projects, slices, and user information (repository/database)
- **Slice Control and Management layer (SCML)**: contains the various aggregate managers which aggregates different resources to give a unified view to the upper layer. The components residing at this layer will enforce policies on the components in the lower layer. In this layer, aggregate managers can be separate entities or can be an aggregate of aggregate managers forming a hierarchy of aggregate managers
- **Resource Management layer**: manages the resources in the OFELIA facility. Irrespective of the virtualization techniques used for different resources, the objective of RML is to provide an experiment with the illusion that it is running on its own dedicated infrastructure. Two resource managers are identified:
  - the RM for OpenFlow-enabled equipment (using FlowVisor)
  - the RM for virtual machines (using XenServer hypervisor technology)

Components across these layers (e.g. hierarchy of aggregate managers) may differ based on the individual offering to the OFELIA facility.

### Federation Mechanism

OFELIA control framework was based on SFA, a federation framework which defines a set of rules by which two or more experimental entities can be federated. The OFELIA facility exists as a federation of heterogeneous experimental facilities with a homogeneous control framework. This is called intra-federation.

### Intra-federation

It follows the same architecture as the single island architecture, and is visualized in Figure A.2. The common problems in an intra-federation experiment are related to identity, authority management and also the control procedures which are inherently handled by the OFELIA control framework. All the available resources are accessible through the control framework. The UI is an entity which talks to the clearinghouses. The clearinghouse/Slice manager is responsible for communicating with all the aggregate managers through its southbound interface to collect all information regarding the available resources and present it to the UI layer.

### Inter-federation

It is defined as the federation of heterogeneous experimental facilities with heterogeneous control frameworks.

In order to support federation with other testbeds OFELIA identifies following requirements that the control framework should fulfil:

- Unified profile for certificate authority management
- Control frameworks that support common interfaces or adapters
- Common data access interfaces

When satisfying inter-federation in OFELIA control framework, the resources will be made available to a different control framework through its association with the aggregated aggregate managers. Aggregate AMs collect all information from the lower level AMs to present the available resources to the other federation. Interior and

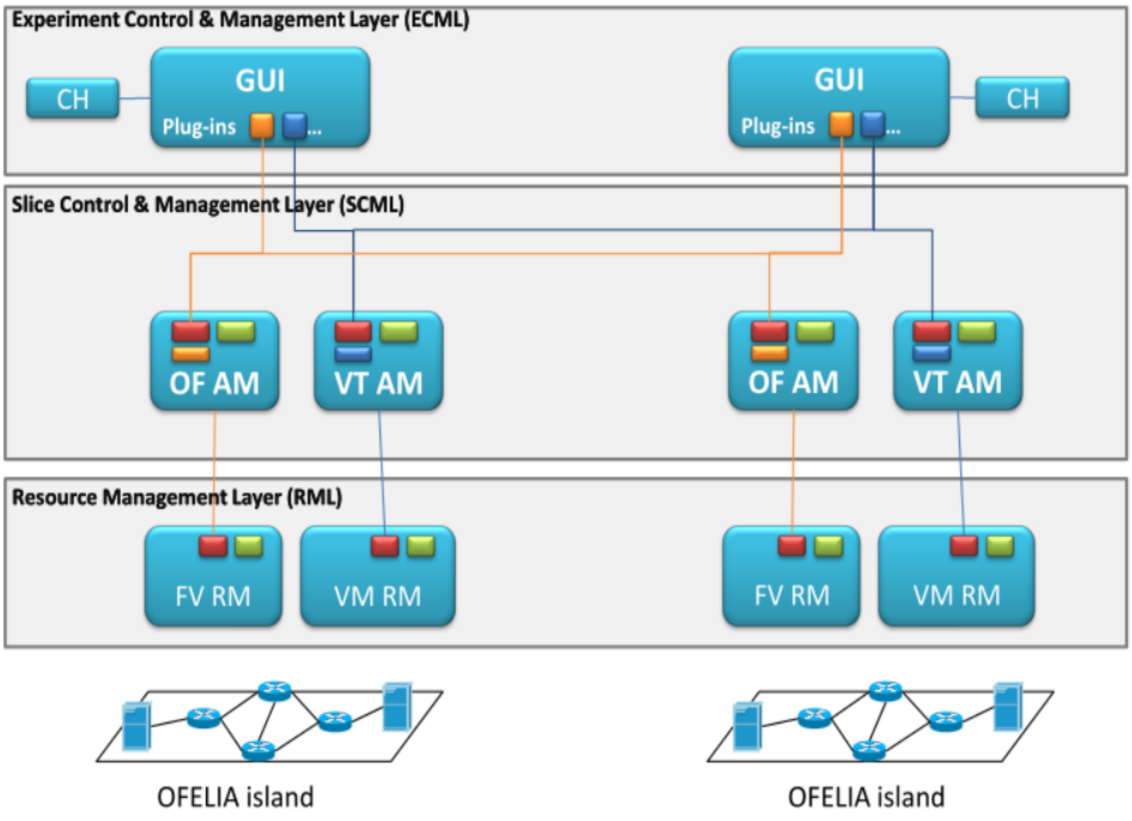
Experiment Control & Management Layer (ECML)
CH
GUI
Plug-ins
...
GUI
Plug-ins
...
CH
Slice Control & Management Layer (SCML)
OF AM
VT AM
OF AM
VT AM
Resource Management Layer (RML)
FV RM
VM RM
FV RM
VM RM
OFELIA island
OFELIA island

Figure A.2: Architecture for intra-federation

exterior gateway managers perform the function of importing and exporting resources from other testbeds as described in the following figure:

In order to support federation with other testbeds OFELIA control framework architecture supports a pluggable environment to add new interfaces. The plugin system has SFA plug-in support to interface with existing SFA based testbeds. It is also modular enough to create new plug-ins as needed.

## FIBRE

All the information in this section is taken from [2], [12] and [13], to which refer for further details.

The FIBRE (Future Internet testbeds and experimentation between BRazil and Europe) project aims to design, implement and validate a shared Future Internet research facility, supporting the joint experimentation of European and Brazilian researchers. This overall main goal can be broken down into the following objectives:

- Build a shared-scale experimental facility.
- Federate the Brazilian and European facilities.
- Showcase the potential of the infrastructure.
- Enhance the collaboration and exchange of knowledge between European and Brazilian researchers in the field of Future Internet.

The FIBRE testbed is a federation of several data-centers distributed across Europe and Brazil managed by different kind of control and monitoring framework (OFELIA, OMF and ProtoGENI). This federation joins three testbeds:

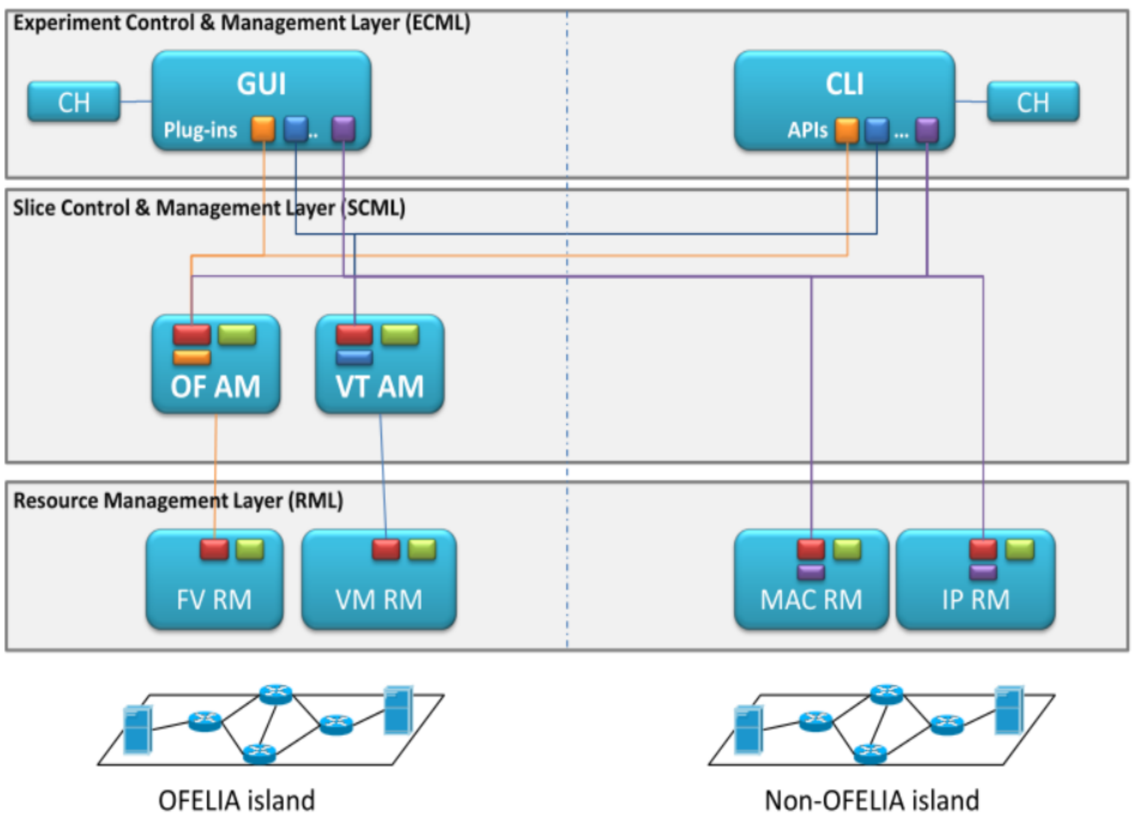
Experiment Control & Management Layer (ECML)
CH
GUI
Plug-ins
CLI
APIs
CH
Slice Control & Management Layer (SCML)
OF AM
VT AM
Resource Management Layer (RML)
FV RM
VM RM
MAC RM
IP RM
OFELIA island
Non-OFELIA island

Figure A.3: Architecture for inter-federation

- Ofelia testbed, based on OpenFlow technology.
- NITOS testbed, wireless nodes based on commercial Wifi cards and Linux open source drivers.
- FIBRE-BR testbed, including nine Brazilian partners interconnected using private L2 channels.

The Figure A.4 shows the overall scenario of FIBRE testbed.

## FIBRE Architecture

The FIBRE architecture (or Federation Environment) is the result of an analysis and internal project decisions related to federation issues on the number of authorities, the naming, the user portals and the federation of the control plane.

**Authorities.** FIBRE testbed is designed to have two top-domain authorities, the first under the responsibility of Brazil and the latter under Europe responsibility. These two authorities will be inter-connected in order to achieve a federation implying that a SFA Register has to be deployed in each side and each authority has to sign the certificate issued by the other authority.

The Figure A.5 shows the peering of EU-BR authorities for FIBRE federation.

**User Portal.** FIBRE testbed is designed to have at least one portal per top-authority. The chosen software component is MySlice tool.

The Figure A.6 shows the integration of MySlice component into FIBRE architecture.

**Federation Control Plane.** The MySlice portal will interact with NITOS (based on OMF) and OFELIA CMFs through its SFA-GW (SFA gateway) component, which includes the responsibility of the slice management. The SFA-GW directly interacts with OFELIA Aggregate Managers which already support the GENI version 3 APIs. On

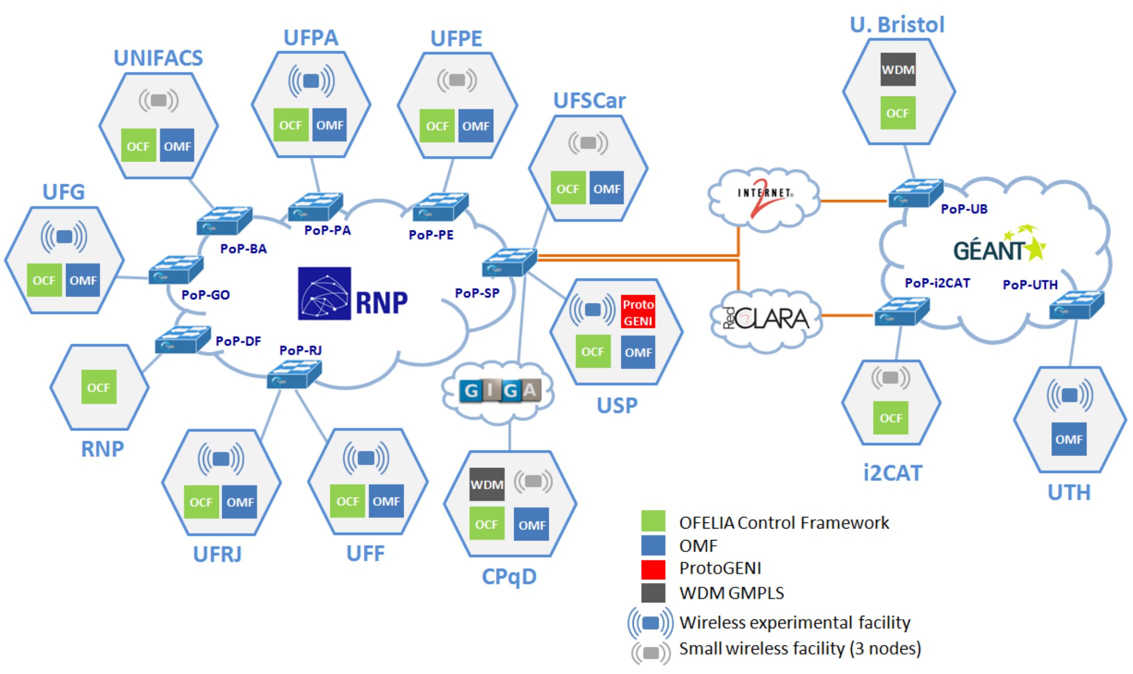

Figure A.4: FIBRE testbed

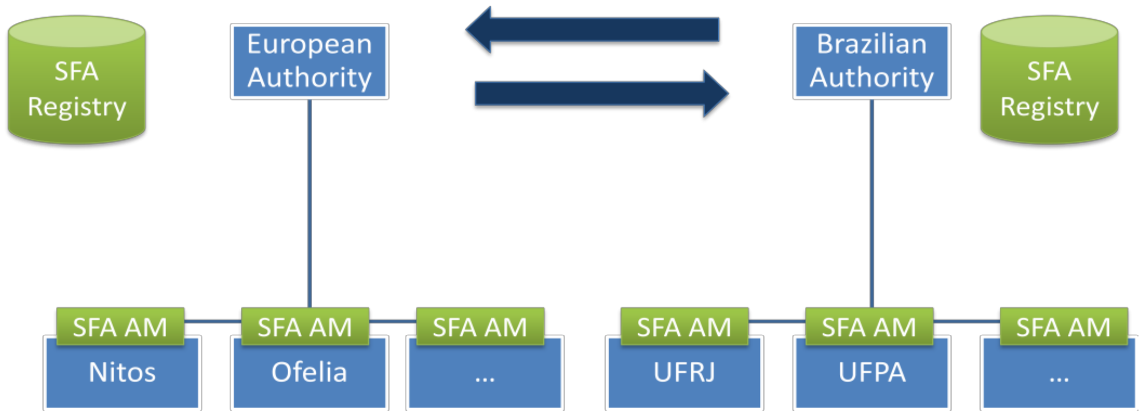

Figure A.5: FIBRE top-authorities

the other hands, the SFA-GW will need an SFA Wrapper (or SFA Driver) to communicate to OMF which still does not support SFA APIs in the version 5.4. It is important to note that OMF 6.0 will natively support SFA through the omf_sfa component.

The Figure A.7 shows the interaction of MySlice portal with the CMFs.

**Synchronization of LDAP and SFA Registry.** The synchronization between LDAP and SFA Registry is a crucial point for FIBRE testbed allowing the reuse of the LDAP user management, already deployed in OFELIA and Brazilian testbed.

## FIBRE Use-Cases

The FIBRE project has identified three scenarios to evaluate the deployed local and federated facilities.

**Seamless mobility for educational laptops.**

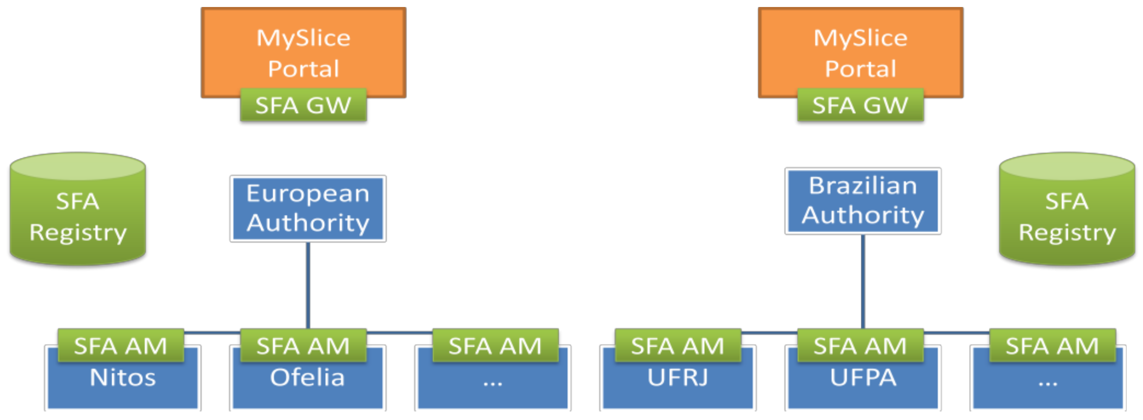

Figure A.6: Fibre federation portal

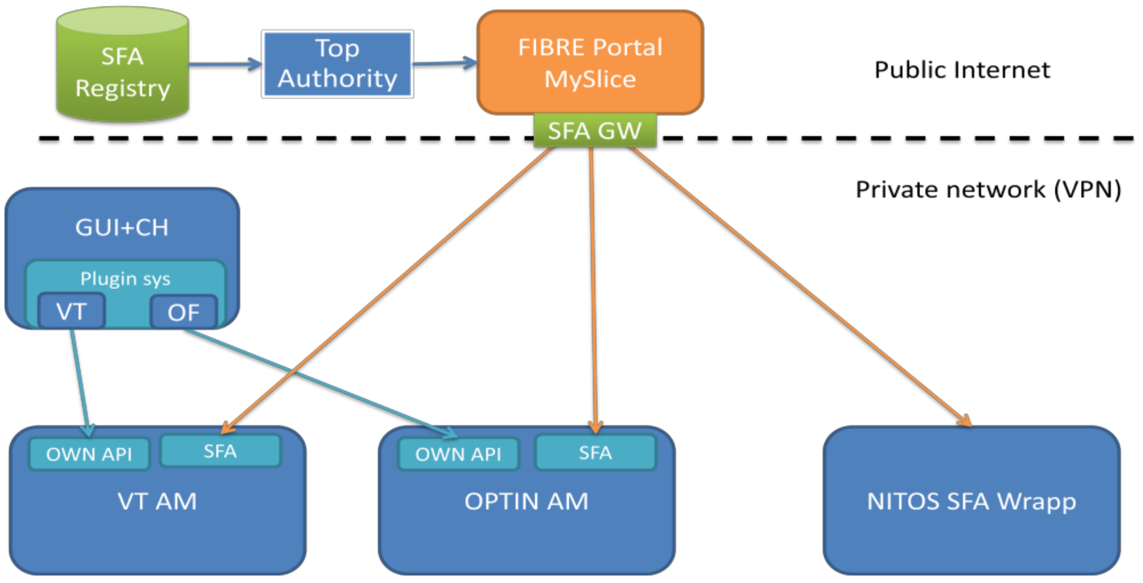

Figure A.7: FIBRE federation control plane

The goal of this use-case is to analyze and utilize the capabilities of wireless networks to augment the seamless handoffs experience on networks formed by mobile users, i.e. people using smart phones, tablets, netbooks, and notebooks. This tech-pilot aims to join the common OpenFlow enabler and WiFi access point providing spatial coverage for experimental wireless communication, laptops and/or programmable handheld devices with WiFi and Bluetooth (BT) interfaces, plus instrumentation for traffic generation and analysis and mobility emulation.

**High definition content delivery across different sites.**

The idea behind this use-case is that an OpenFlow based application (i.e. a NOX application) can be interfaced to one or more Content Delivery Servers (CDSs) that form a Content Delivery Network (CDN). This application could monitor the CDS performance by retrieving the related status, load and failures. When certain thresholds are exceeded (e.g. the load on CDS or its energy consumption), NOX application can re-route one or more clients to another CDS located in another site. The re-routing can be performed and facilitated by NOX application which can easily change the flow tables of the OF switches under its control.

**Bandwidth on demand through OpenFlow GMPLS in the FIBRE facility.**

This use-case aims to analyze the flexibility of the OpenFlow protocol and NOX control platform in a close collaboration with a GMPLS PCE (Path Compute Element) module to implement an open and generalized Bandwidth on Demand (BoD) service on virtualized networks. The Figure A.8 shows the overall architecture with the design modules and their interfaces:

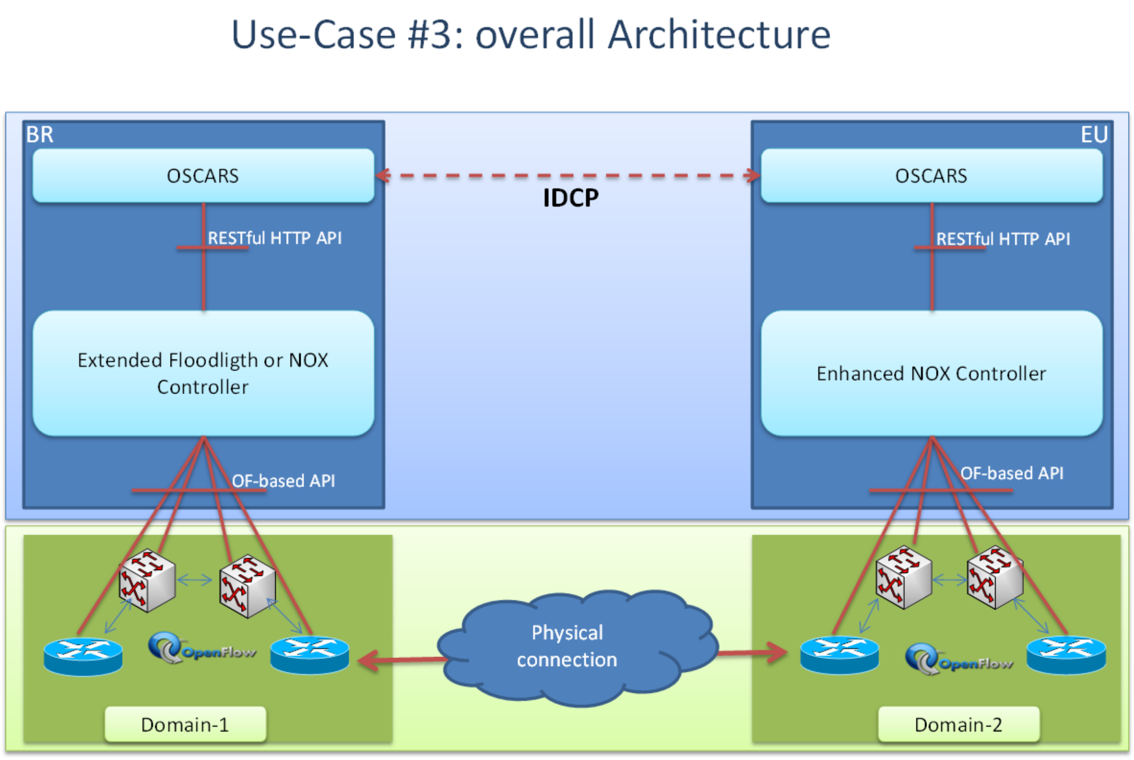

Figure A.8: Fibre UseCase 3 overall architecture

Several building blocks are deployed to fulfil the requirements:

- Flowvisor, used to virtualize the physical network topology.
- Ofelia Control Framework, used to manage each island's resources (Openflow switches and Virtual machines).
- NOX controller, used to retrieve the network information and to create/destroy the flow-entries into the OpenFlow enabled switches.
- Flow-Aware PCE, used to calculate a path between source and destination end-points.
- OSCARS, used to perform the BoD requests and to share topology details between different islands.

The Figure A.9 depicts the physical interconnection between a Brazilian partner (CpQD) and an European partner (i2CAT) through a VPN Layer 2 circuit.

## Fed4FIRE

In order to achieve the desired architecture for resource discovery, requirement, reservation and provisioning, the following components must be deployed within a federation facility:

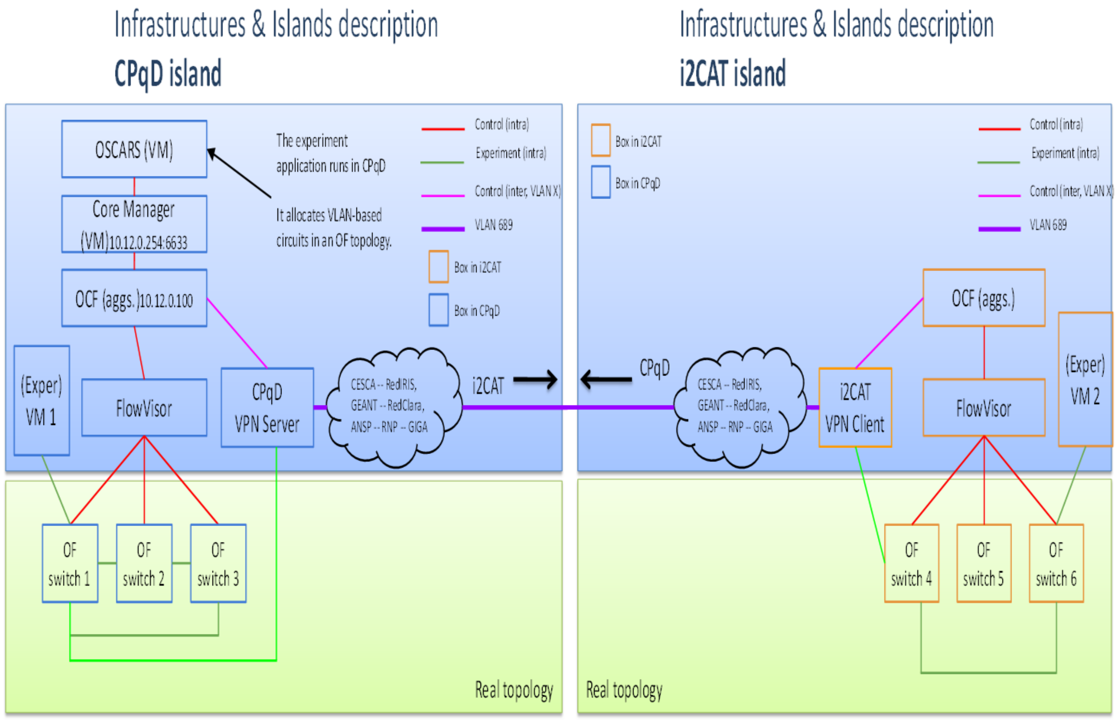

Figure A.9: Fibre UseCase 3 Physical Interconnection between Brazil ans Europe

- Portal: A central starting place for (new) experimenters. Totally new experimenters can register on the portal. The portal will also provide a view on the available resources in the federation, supporting resource discovery, requirements definition, reservation and provision operations. Note that, besides, other experimenter standalone tools can provide the same functionality but novel experimenters will be able to access through a single site.

- Identity provider: Experimenters identify themselves at the portal as federation experimenters are registered at this identity provider. As can be seen in the figure, testbeds can also deploy their own identity provider (testbed A). However, if testbeds do not want to go through the burden of setting up their own identity provider, they have to outsource this functionality to the identity provider of the Federation Facilitator.

- A testbed directory: A directory readable by humans and by computers that has an overview of all testbeds in the federation. In its computer readable form, the testbed directory is merely a listing of the IP addresses corresponding with the different testbed management software deployments that expose the common interface for discovery, reservation and provisioning. In its human readable form, this directory is a webpage that displays some introductory information about each testbed belonging to the federation.

- A tool directory: It gives an overview of available tools for the experimenter. This will again be a webpage were more information regarding FIRE tools is gathered. This can cover both tools that are officially endorsed by some of the testbeds, and other tools that are brought to light by experimenters or the tool developers. Users will gain access to the human readable testbed directory through the portal.

- Certificate directory: In our distributed architecture, the targeted chain of trust implies that testbeds should import the root certificates of the different identity providers present in the federation into their own access control components. This way the testbeds can verify that an incoming user is indeed affiliated with a federation member. These root certificates could just be manually exchanged between the different testbed operators. However, it is more convenient to provide a trusted location that makes all the federation root certificates available in a single download. This is the goal of the certificate directory.

- Future reservation broker: A tool to facilitate future reservations of resources, this broker can help to find and subsequently reserve the right time slots and resources over multiple testbeds. A single query to this broker will provide the information for all testbeds.

A Fed4FIRE testbed has the following attributes:

- A testbed may be or may not be an identity provider.
- For authentication/authorization between users and testbeds, a trust model is used. Identity providers trust each other and specific experimenter properties are included in the experimenter's certificate, which is signed by the identity provider. Testbeds can therefore do rule-based authorization. This means that incoming users cannot only be distinguished based on the affiliation, but also on the experimenter's profile. For instance, because of this approach, a testbed is able to define and enforce the policy that experimenters coming in from the federation should at least have reached the Ph.D. student level. This means that on this particular testbed, master students will not be granted access, even if they are affiliated with a member of the federation.
- A testbed can query/trust the central certificate directory to see which root certificates it should trust. This is more convenient than manually retrieving all certificates from the different identity providers within the federation.

The existing component(s) responsible for discovery, reservation and provisioning should expose this functionality through a common interface. SFA is considered to be a suitable choice for such a common interface. However, when adopting SFA for the envisaged heterogeneous federation, three important additions are needed:

- 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 to further explore the use of ontology based descriptions for these RSpecs in the context of the Fed4FIRE project. This should make it easier for experimenters, experimenter tool and broker developers to use these resources.
- Policy based authentication is considered to be very important. Credentials, certificates and policy engines should be extended as such.
- There is no concept of future reservation in the AM API at this moment. This extension will be studied further.

## Monitoring and Measurement

The following types of monitoring and measurement are identified in Figure A.10:

- Facility monitoring: This provides monitoring information 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). Another possibility is to expose the information already registered by currently deployed monitoring tools. In both cases, the interface on top of this facility monitoring should be the same. It needs further specifications.

- Infrastructure monitoring: This provides monitoring information about the infrastructure resources that is useful for experimenters. For instance, providing measurement data about resources such as 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 for instance no access to the physical host if he uses virtual machines), and as such, a common interface is needed but is not existing today .
- 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. As illustrated in Error: Reference source not found for instance, 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-installed.

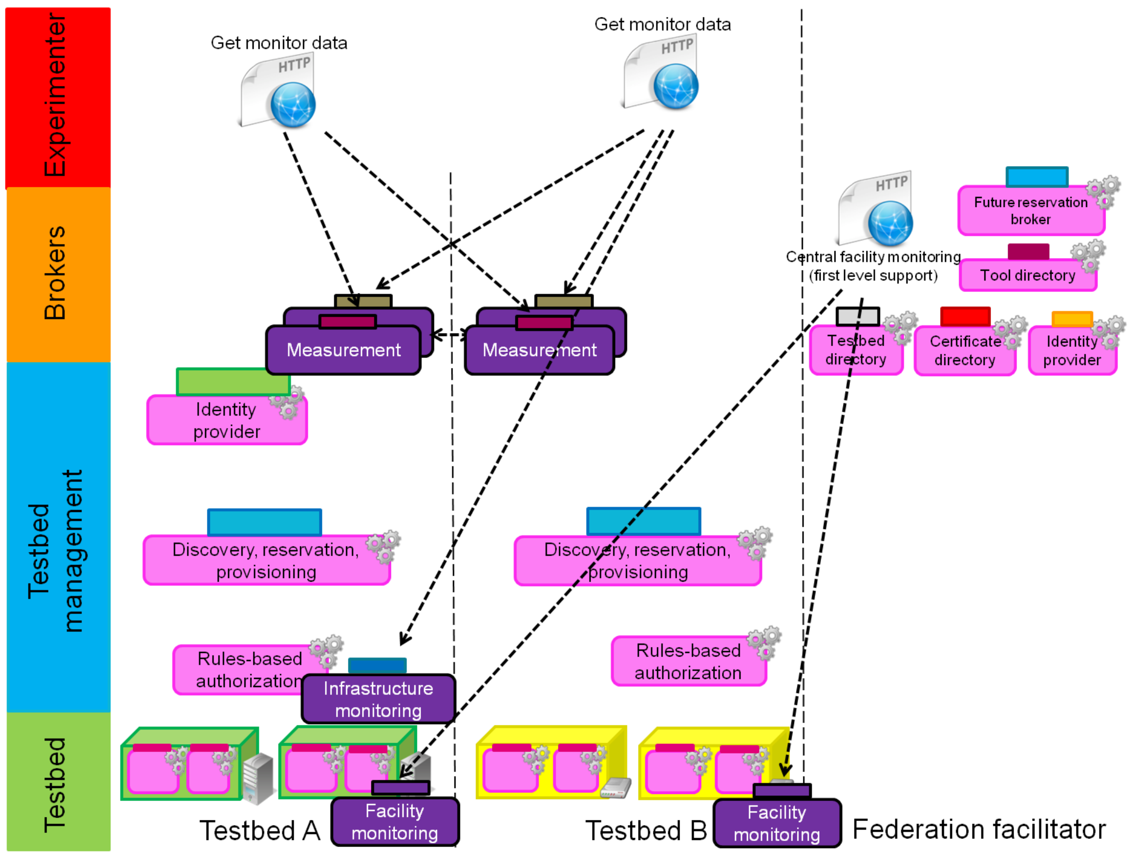

Figure A.10: Monitoring and measurement architecture

## 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. So from an architectural point of view, no specific components need to be introduced to support experiment control. However the testbed providers could ease this by putting certain frameworks pre-installed in certain available disk images. Ideally, these frameworks are based on a standard resource control protocol, since this will permit to control resources provided by federated facilities using different management software in a uniform way. An example of such a protocol is the federated resource control protocol (FRCP) .

## BonFIRE

All the information in this section is taken from [1], [18] and [19], to which refer for further details.

The BonFIRE (Building service testbeds for Future Internet Research and Experimentation) project goal is to provide a state-of-the-art multi-site cloud facility for applications, services and systems research in the Internet of Services (IoS) community. This facility can give researchers access to large-scale virtualized compute, storage and networking resources with the necessary control and monitoring services for detailed experimentation of their systems and applications.

BonFIRE comprises 7 geographically distributed testbeds across Europe, which offer heterogeneous cloud resources, including compute, storage and networking. The Figure A.11 shows further details about resources offering on the different testbeds.

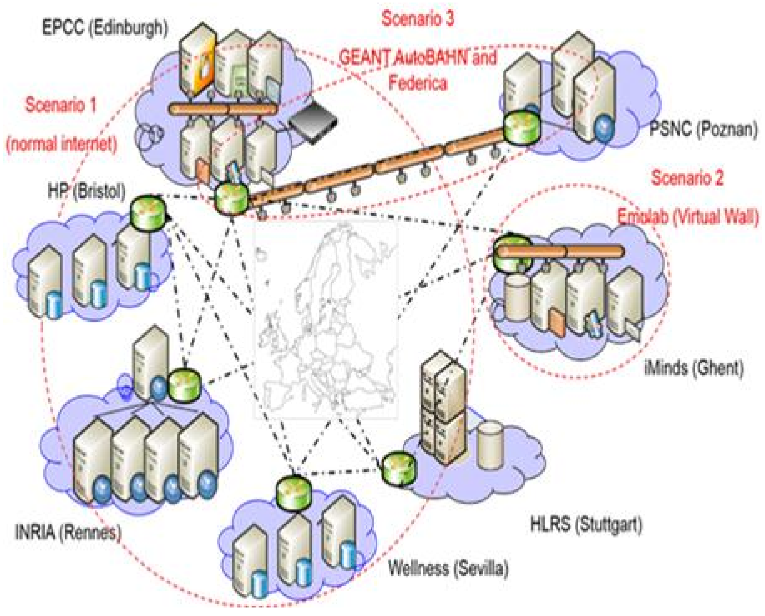

Figure A.11: Geographically distributed testbeds

**Resource control.**

BonFIRE offers the complete control of compute, storage and networking resources. It supports dynamically creating, updating, reading and deleting resources throughout the lifetime of an experiment. Compute resources can be automatically configured with application-specific contextualization information.

The BonFIRE framework can offer "on-request" compute resources, allowing experimenters to reserve large quantities of physical hardware (162 nodes/1800 cores available) and giving experimenters flexibility to perform large-scale experimentation.

**Managed experiment environment.**

BonFIRE gives the fully control of the running experiments. An experiment can define the entire infrastructure across all testbeds in a single file descriptor and can perform further complex actions on the newly created infrastructure. E.g.:

- Saving compute disk images with owner software stack or storage resources.
- Sharing saved compute and storage resources.
- Sharing access to experiments with colleagues.
- Repeating experiments and share experiment descriptions for others to set up.

- Aggregated monitoring metrics at both resource level (e.g., CPU usage, packet delay, etc.) and application level.
- Aggregated monitoring metrics at infrastructure level at selected testbeds.

**Ease of use.**
BonFIRE gives a deep control of resources to configure, execute and manage the experiment trying to make this actions as easy as possible. It offers several tools to interact with BonFIRE API, e.g.:

- BonFIRE Portal GUI, where it is possible to create an experiment in a step-by-step manner
- CLI (command line tool), such as Restfully
- A script file descriptor (JSON or OVF based format), which can be automatically executed by Restfully
- Raw HTTP commands via cURL

**What can be tested and how.**
BonFIRE supports experiments exploring the interactions between novel service and network infrastructures. Three initial scenarios have been defined to highlight the general classes of experiment that can be supported by the facility. These scenarios include:

- *Extended cloud scenario*: a federated facility with heterogeneous virtualized resources and best-effort Internet interconnectivity.
- *Cloud with emulated network implications*: experimental network emulation platform under full control of the experimenter
- *Extended cloud with complex physical network implications*: experimental cloud system federated with GÉANT BoD and FEDERICA (collaboration with NOVI)

| **BonFIRE site facility** | **Connecting NREN** |
|---|---|
| EPCC | JANET |
| HLRS | DFN |
| HPLabs | JANET |
| IBBT | BelNET |
| INRIA | RENATER |
| PSNC | PIONIER |

Table A.1: BonFIRE sites and correlated testbeds

## BonFIRE Architecture

The BonFIRE architecture is composed by several layers (Portal, Experiment Management, Resource Management, Enactor and Testbed site layer) and a set of cross-cutting capabilities for monitoring and identity management. Each layer exposes functionalities via a set of well-defined APIs.
The Figure A.12 shows a high-level overview of the BonFIRE architecture.

- **The Portal.** The Portal offers the experimenters a graphical user interface showing the running experiments, the available resources at each testbed site, the monitoring information, etc…
- **The Experiment Manager.** The Experiment Manager provides an interface to schedule, plan and orchestrate the execution of an experiment as described by a file descriptor (with all resources for an initial

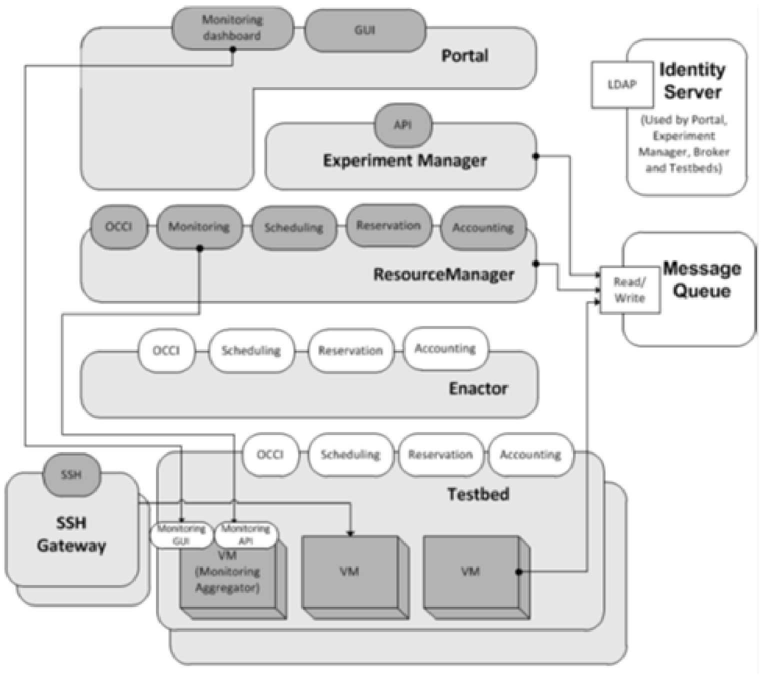

Figure A.12: BonFIRE Architecture

deployment).

- **The Resource Manager.** The Resource Manager provides an interface to create, manage and terminate compute, storage and network resources, which may physically reside at any testbed in the BonFIRE system.

- **The Enactor.** The Enactor allows the decoupling of the specific implementations of the testbed API from the BonFIRE Resource Manager providing a well-defined and “standardized” interface (like a driver or a general plugin).

- **The Testbed Sites.** The BonFIRE cloud testbed sites use a common OCCI interface to expose resources (compute, storage and network) to the Enactor. Compute resources refers to virtual machines created for experiment, network resources connect these VMs and storage resources are disk-blocks attached to VMs.

### Cloud-to-Network Interface

BonFIRE provides the experimenters with the possibility to request QoS-enabled network connectivity services with a guaranteed bandwidth to interconnect BonFIRE sites (Bandwidth on Demand services). Instead of relying on the best-effort Internet connectivity, the cloud resources located in different BonFIRE sites can be interconnected through a dedicated network service with the bandwidth requested by the experimenter in terms of *minimum bandwidth reserved* and *maximum bandwidth guarantee*.

The **inter-site** BoD services are provided by a third-party network provider connecting the BonFIRE sites. In particular, BonFIRE adopts the GÉANT Bandwidth-on-Demand system (AutoBAHN), which is a mature solution in terms of specifications, implementation and deployment in a multi-domain environment.

The Figure A.13 depicts two of BonFIRE sites (EPCC and PSNC) connected to the European GÉANT network.

In order to integrate the AutoBAHN services, BonFIRE architecture was enhanced with a new module (AutoBAHN Adaptor) and a new resource type (Site-Link). The AutoBAHN Adaptor's main functionality is to translate BonFIRE OCCI to BoD service requests, basically SOAP requests towards the AutoBAHN User Access Point (UAP)

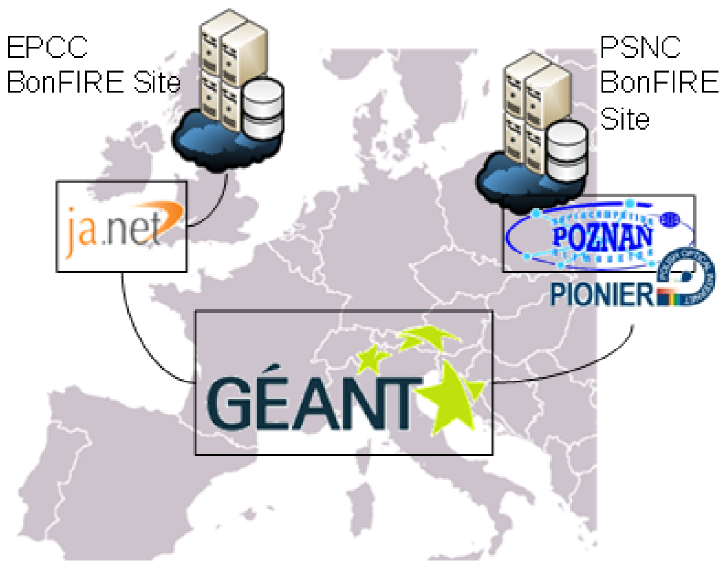

Figure A.13: BonFIRE Cloud-to-Network interface

interface. The Site-Link resource is a new type of OCCI resource (like storage or network resource), which can be manipulated following the common CRUD mechanisms adopted in BonFIRE to create, remove and query resources via OCCI interfaces. The site-link resource is described by several network parameters, i.e. *endpoints* (each BonFIRE site connected to GÉANT), *vlan identifier*, *bandwidth constraint*, etc..

The Figure A.14 shows the enhancement to BonFIRE architecture with AutoBAHN services.

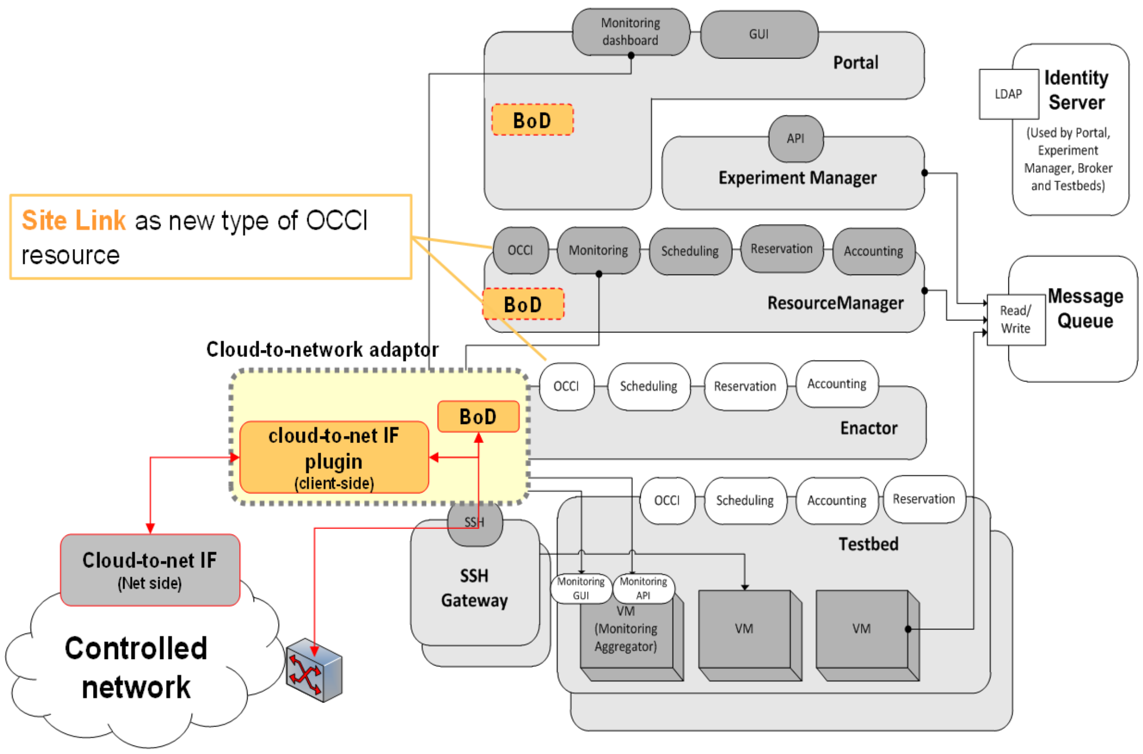

Figure A.14: Enhancement to BonFIRE architecture with Cloud-to-Net extensions

## GridARS

The GridARS (Grid Advanced Resource Management System) framework is a reference implementation of the Open Grid Forum (OGF) Network Services Interfaces (NSI), Connection Service (CS) protocol standard, developed by AIST. The CS protocol version 2 is a Web services-based interface to reserve, provision, release and terminate a service, such as a end-to-end connection, via a two-phase commit protocol. GridARS can coordinate multiple resources (services), such as a network connection, virtual machines and storage spaces, via the CS protocol.

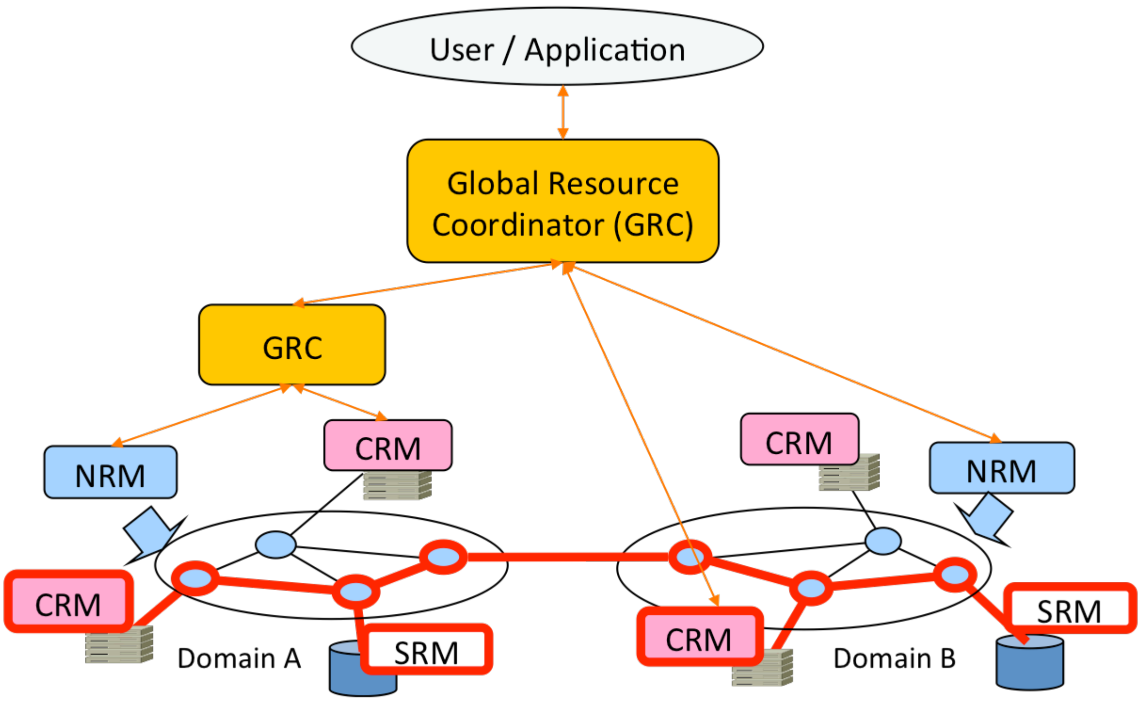

Figure A.15: GridARS resource management configuration.

Figure A.15 shows a resource management configuration assumed by GridARS. In Figure A.15, Domain A and B denote network domains managed by different administrative organizations. This resource management configuration consists of a Global Resource Coordinator (GRC), which coordinates heterogeneous resources, and Resource Managers (RMs), which manage each local resource directly. GRCs and RMs can work together to provide users a virtual infrastructure over multiple domain physical resources. NRM, CRM and SRM in Figure A.15 denote RMs for networks, computers and storage, respectively. More than one GRC is allowed in a single system. GRCs could be configured in a coordinated hierarchical manner, or in parallel, where several GRCs compete for resources with each other on behalf of their requesters, such as users and applications.

### GridARS Architecture

Figure A.16 illustrates GridARS architecture. In order to provide requesters with a virtual infrastructure, which spans several cloud resources, provided by multiple management domains including commercial sectors, GridARS provides three service components:

- Resource Management Service (RMS)
- Distributed Monitoring Service (DMS)
- Resource Discovery Service (RDS)

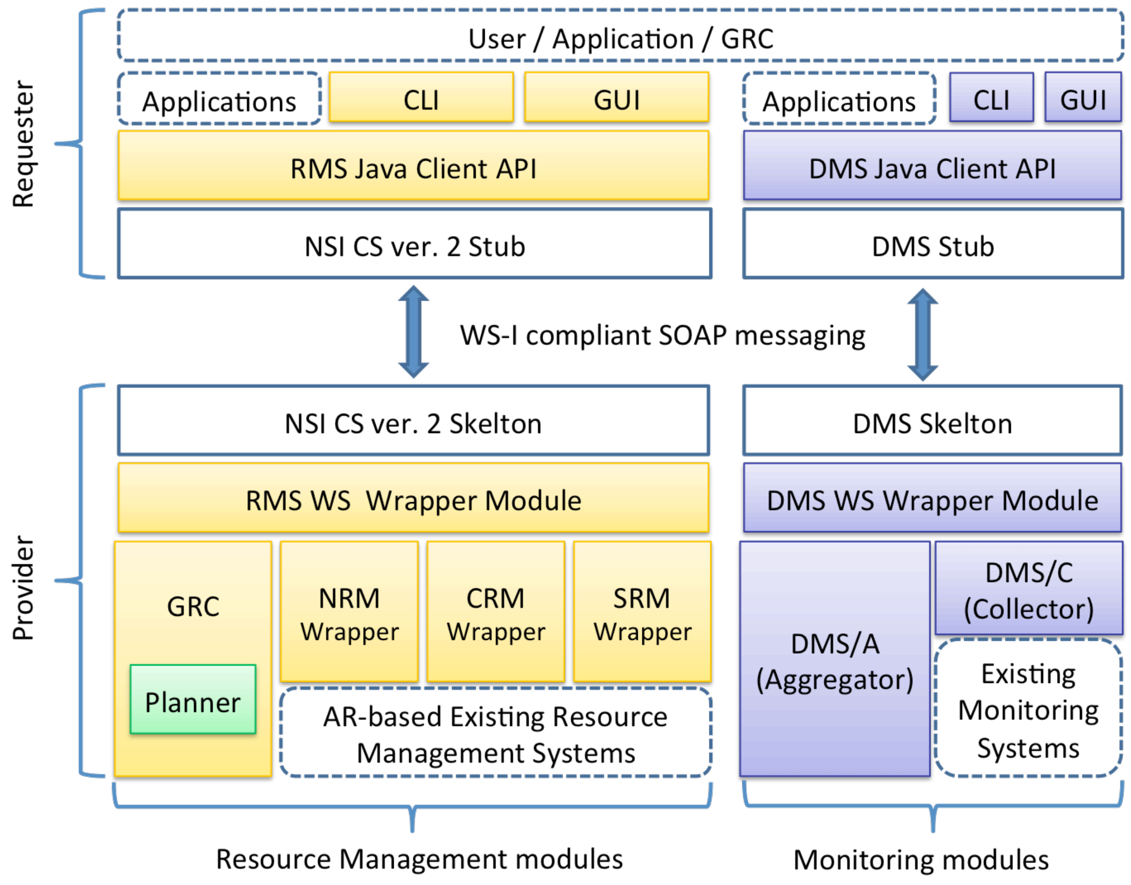

Figure A.16: GridARS service components.

Resource Management Service (RMS) is based on NSI CS and consists of GRCs and RMs. Co-working with GRCs and RMs, RMS enables to coordinate heterogeneous virtual resources on multiple cloud environment. Here, a virtual resource means a part of physical resources, sliced and isolated by other users or applications. For example, a network resource is an end-to-end bandwidth guaranteed connection and its detailed physical topology does not need to be disclosed. GRC has a co-allocation planning capability, called Planner. Planner determines a suitable resource allocation plan. Based on the allocation plans, GRC can perform resource reservation on subordinate GRCs or RMs.

Distributed Monitoring Service (DMS) allows to the requesters to monitor the virtual environment, allocated to them. DMS does not have a central database, such as MonALISA and PerfSONAR, and each virtual resource usage is monitored, managed and filtered by each cloud administrator. DMS gathers such distributed monitoring information, tracking the hierarchical RMS reservation tree using the reservation ID, automatically. DMS consists of Aggregators (DMS/A) and Collectors (DMS/C). DMS/A gathers monitoring information from related DMS/As or DMS/Cs distributed over multiple domains, and provides the information to the requester. Each DMS/C monitors the reserved resources periodically, filters the monitoring information by the domain policy, and provides the requester with the authorized information. Based on the monitoring information, the requesters can recompose the virtual infrastructure for their applications.

Resource Discovery Service (RDS) collects static resource information items from each resource domain, and provides the aggregated information. The RDS implementation is based on Catalog Service Web (CSW), defined by Open Geospatial Consortium (OGC), which is an online XML-based database. Each resource domain can POST an XML document, which describes its static resource information, such as network topology, number of VMs, and storage spaces.

## RISE

Since 2009, JGN-X have been working on developing a nation-wide OpenFlow testbed "RISE(Research Infrastructure for large-Scale network Experiments)". RISE project is successfully running an OpenFlow testbed over JGN-X, with fully utilizing its wide-area coverage from US West coast to Southeast Asia. RISE provides the wide-area OF network composed by the hardware OpenFlow switches, also provides the RISE OF Controller based on Trema which developed by NEC. Also researchers and developers can try their own OF controller on the RISE network for their experiment. And some SDN or Cloud developers can try their own software examination with the dynamic network provisioned by the RISE controller. Currently, RISE has 11 sites in Japan, and three sites in overseas. For each site, RISE has a few OpenFlow switches and two VM servers (Japan domestic only). Currently, there is no any control framework and portal. It means FELIX control framework will be great contribution to them.

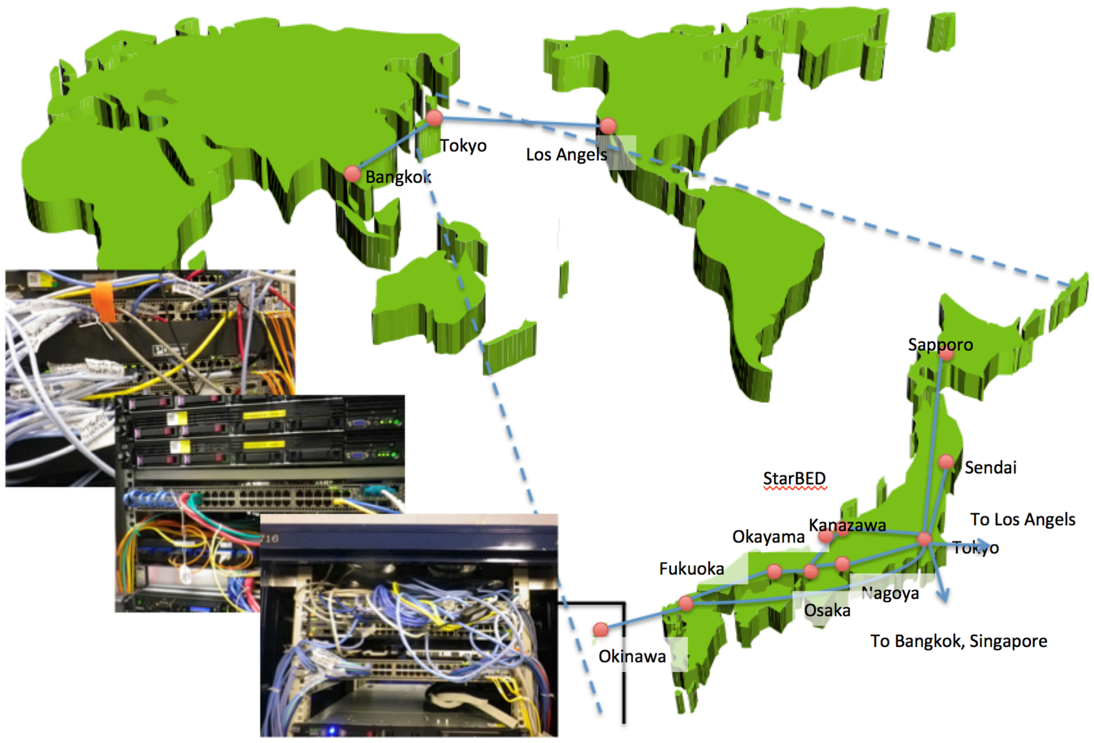

Figure A.17: Global RISE testbed infrastructure

### Previous RISE Architecture

RISE is constructed over the JGN-X network, thus links between OpenFlow switches are implemented by traditional VLAN technology. In order to create the network slices, RISE divided one physical OpenFlow switch into 16 VSIs (Virtual Switch Instance). This virtualization mechanism is not provided by OpenFlow standard, but OpenFlow switch specific function. Therefore, 16 users can share single physical OpenFlow switch at maximum. In addition, VM servers are installed on each site and VMs are attached to the experimental user's slices.

For the link between OpenFlow switches, it's implemented by traditional VLAN technology. Therefore, it was difficult to isolate "RISE user slice" and another user's slice created by only VLAN. So they decided to use Pseudo Wire technology for data-plane between OF switches in each site. However, they had the issue on the topology. They have ten sites in Japan and three sites in overseas (Los Angeles, Bangkok, Singapore) but its topology is not

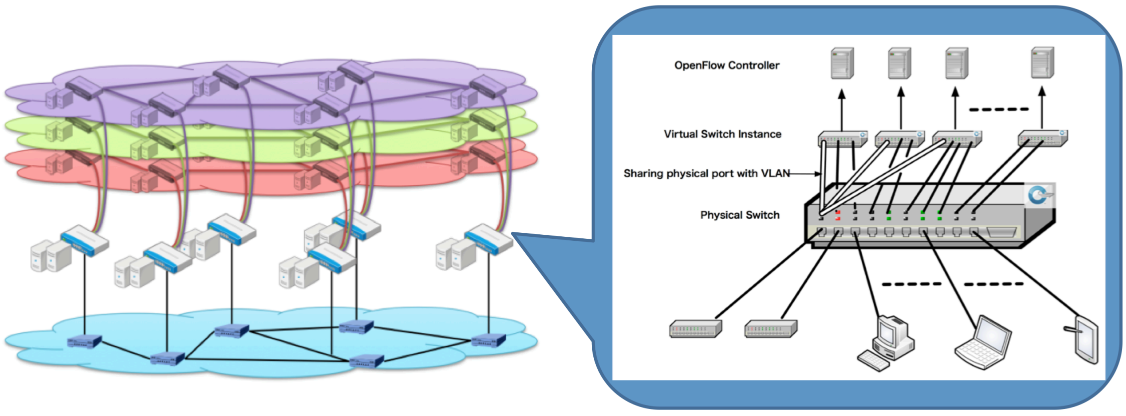

Figure A.18: RISE Architecture

full mesh. As the result, users concentrate sites which can create the loop topology, then fully-utilized the testbed environment. In order to resolve this issue, they develop the new architecture called RISE3.0.

### Current RISE Architecture

In this RISE 3.0 design, RISE team employs logical path system because it is easy to implement. This method replaces ethernet switch which locates between user OpenFlow switch and JGN- X switch, with OpenFlow switch. It called this OpenFlow switch "RISE OFS" (Figure A.19: RISE Logical path). In RISE 3.0, they provide logical neighbor link for user OpenFlow switches combining physical neighbour links. In Figure A.19, user OpenFlow switches Ua, Ub, Uc connect RISE OFSes Ra , Rb , Rc respectively. We assign ports to logical paths for each users. Ra , Rb , Rc connect JGN-X switches Ja , Jb , Jc respectively, and JGN-X switches forward packet whose VLAN is already configured, with Pseudo Wire . In this example, VLAN ID "L" is assigned to link between Ja and Jb, "M" is assigned to link between Jb and Jc. And logical path "R" is configured between Ua and Uc. Path "R" consists of multiple physical links "L" and "M". Logical path is defined by user identifier and edge ports of RISE OFS.

### Implementation of Logical Path

For implementation of RISE logical path, there are following two methods.

- 1) VLAN stacking

This method uses VLAN ID as logical path identifier. For each logical path, it assigns VLAN ID (hereafter, logical path ID), then RISE OFS fowards packets with the logical path ID. Hereafter, we express Pac as logical path ID. The packet from Ua to path "R" is added logical path identifier "R" at Ra. Then, to forward neighbour JGN-X ethernet switch Jb, Ra adds VLAN ID "L" according to physical link Lab which consists of logical path "R". In other words, controller adds flow entry to add VLAN IDs "R", "L" in order, and to forward to Ja from OpenFlow switch Ua port for path "R". In Rb, it trims top VLAN ID "L", then it observes "R". Rb recognizes that it is on the way of "R", thus adds new neighbor link ID "M" to foward next JGN-X ethernet switch Jc. Finally, Jc receives the packet, sends to Rc, Rc removes VLAN ID "M", "R", and forwards Uc. As explains above, we can implement logical path stacking VLAN IDs which specify logical path and neighbour link. RISE OFS is able to decide the destination of receiving packet according to VLAN IDs.

- 2) Address rewriting

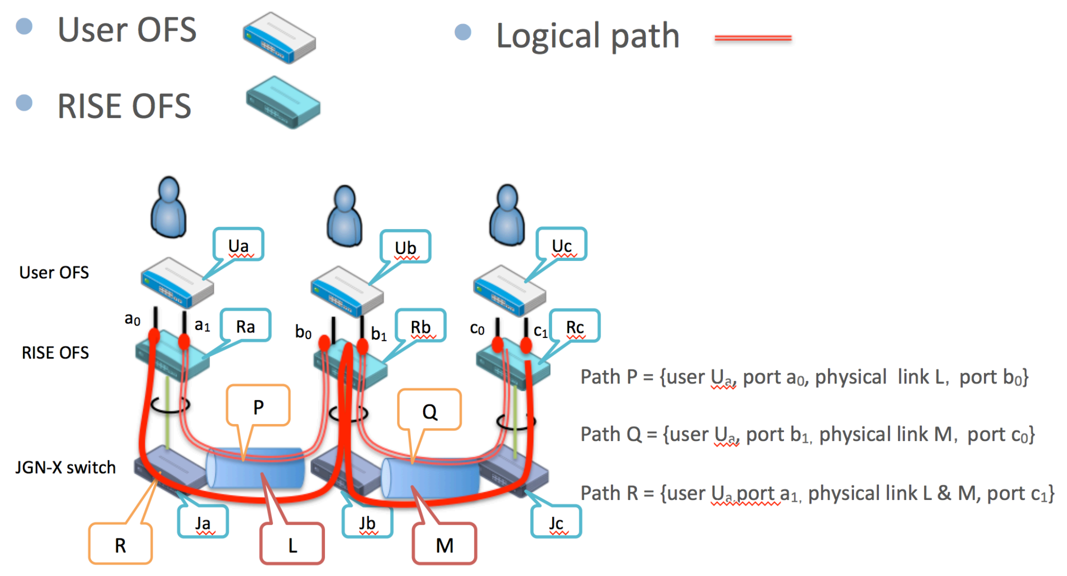
User OFS
RISE OFS
Logical path
User OFS
RISE OFS
JGN-X switch
Ua
Ub
Uc
Ra
Rb
Rc
a0
a1
b0
b1
c0
c1
P
Q
R
L
M
Ja
Jb
Jc
Path P = {user Ua, port a0, physical link L, port b0}
Path Q = {user Ua, port b1, physical link M, port c0}
Path R = {user Ua port a1, physical link L & M, port c1}

Figure A.19: RISE Logical path

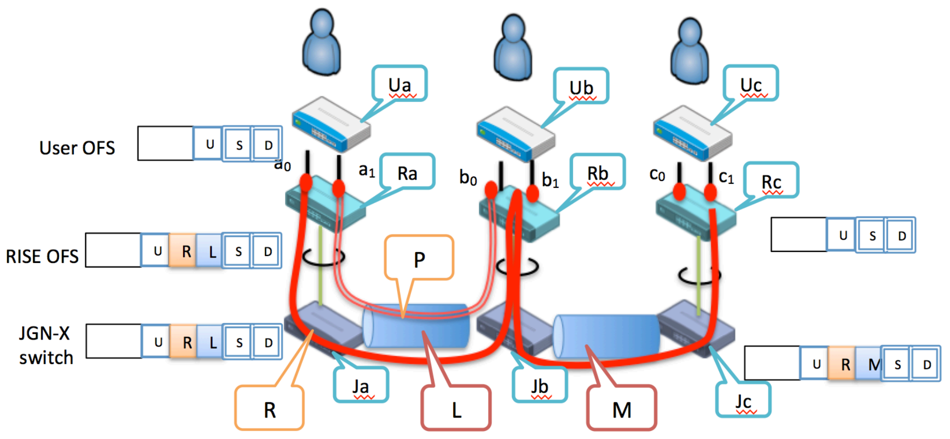

Figure A.20: Implementation by VLAN stack

This method rewrites MAC address of packet to foward along the logical path. In this method, RISE OFS identifies logical path with MAC address instead of VLAN ID. As same as Figure A.19, user OpenFlow switches (except for Ub), RISE OFS, JGN-X ethernet switches are connected. Let logical path be defined by port a0 of Ra and port c1 of Rc. In Figure A.21, Ra and Rc are edge OFSes, Rb is core OFS. When Ra receives packet from Ua to path, it recognizes that the packet belongs to logical path "R" from user identifier U and input port a0. Then, Ra rewrites VLAN ID to "L" to forward the packet to Jb. Finally, it rewrites source and destination MAC addresses to s’,d’ respectively, and sends to Ja. This methods differs from 1) that it does not stack VLAN ID, but rewrites MAC addresses to gurantee uniquness of packet on the logical path. For example, if Ra receives packet whose source and destination addresses are s,d from Ua, it send RISE 3.0 controller Packet-In message. The controller identifies user identifier U and input port a0 from the Packet-In message. The controller can calculate edge OFS Rc and port c1 of the path according to U and a0. Then, it sends flow entries to Ra , Rb , Rc along the path. On Ra, Rc (edge OFS) rewrite VLAN ID and MAC addresses, and on Rb just rewrites VLAN ID ("L"→ "M").

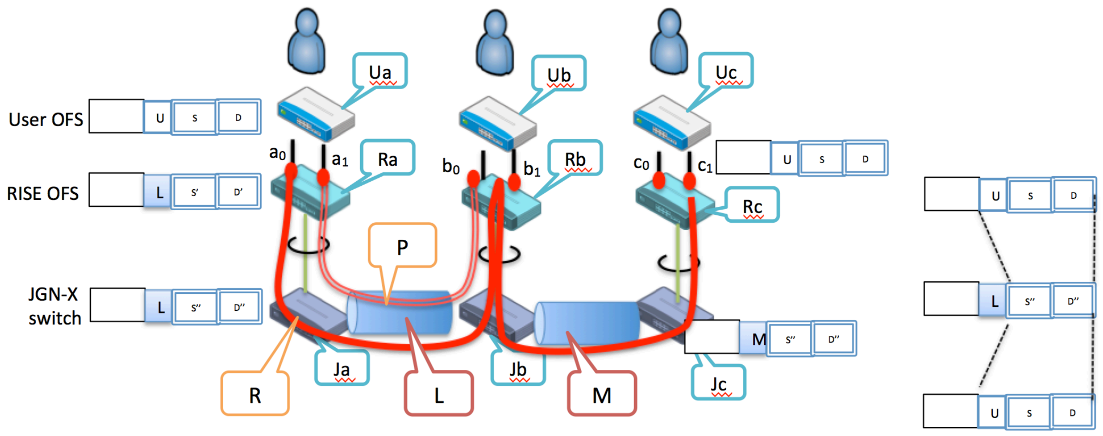

Figure A.21: Implementation by rewriting MAC addresses

RISE 3.0 controller will employ address rewriting method which described in 2). Because of implementation limita- tion of RISE OFS. Because there is a limitation of RISE OFS product specification. In fact, when push two VLAN-IDs to OFS simultaneously, the performance degrades. And also, VLAN stacking is not allowed on JGN-X switches, because of operation policy.

### RISE Use-Cases

Dynamic path provisioning over inter-domain SDN testbed. This use case is aiming to interconnect SDN testbeds in 3 continents and provision the layer2 path dynamically. IDCP was used for the interconnection between domains. To make a provision the path in RISE, we embedded RISE OF controller into OESS, developed by GlobalNOC and deployed in Internet2. We performed dynamic path provisioning demo between RISE SDN testbed in Japan and US Internet2 AL2S in 2013.

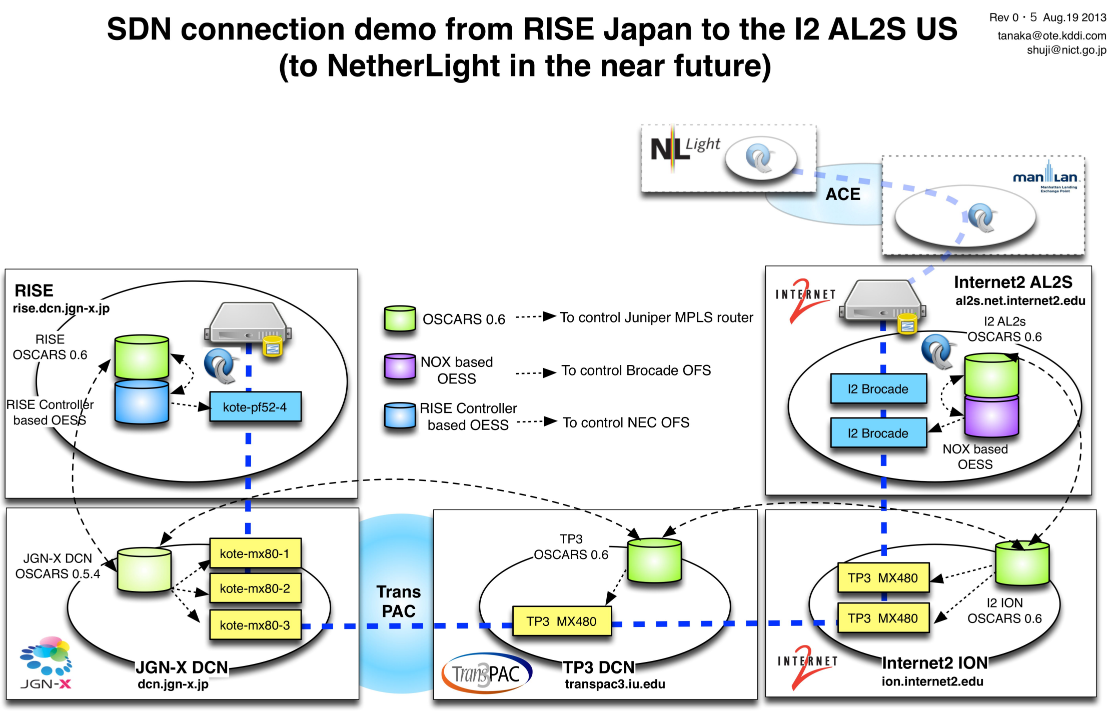

Figure A.22: SDN connection demo

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