The FELIX SDN Experimental Facility

U. Toseef*, K. Pentikousis*, C. Fernandez†, C. Bermudo†, G. Carrozzo‡, R. Monno‡, B. Belter§,
K. Dombek§, L. Ogrodowczyk§, T. Kudoh¶, A. Takefusa¶, J. Haga¶, J. Tanaka‖
*EICT, †i2CAT, ‡NXW, §PSNC, ¶AIST, ‖KDDI
Corresponding author: [umar.toseef]@eict.de

***Abstract*—The development of test environment closet possible to the real world scenario is becoming a fundamental requirement in designing innovative network applications. This environment must be fully configurable and reliable enough to provide the similar results in multiple experiment runs. The federation of the existing Future Internet (FI) testbeds is an initiative to fulfill these strict requirements. The FELIX project aims to define and deploy a control and monitoring framework which allows the experimenters to execute their network services in a distributed environment spread across two continents, i.e. Europe and Asia. This paper describes the architecture of the software components developed to manage heterogeneous resources that constitute the FELIX infrastructure, i.e computing, SDN and transport resources. Due to a modular nature of this architecture, the article introduces the components with particular emphasis to the provided functionalities, the exported interfaces, the dependencies and the relationship between the internal building blocks. Some details of the implementation choices and the workflow to realize the user requests are also presented.**

I. Introduction

In recent years, several projects in Europe, Asia and the US have made significant effort on creating experimental research infrastructures that are reusable and can incorporate the latest emerging network technologies. As a result, a number of SDN test-beds are being developed under the Future Internet (FI) initiative. An open federation of large-scale heterogeneous test-beds is a non-trivial task, requiring specialized design of system architecture and framework. Taking on this challenge, the FELIX project aims to facilitate the federation [1] and integration of different network and computing resources residing in a multi-domain heterogeneous environment across two continents. The FELIX architecture [2] advances the state of the art designs of the SDN test-beds developed in relevant FI research projects thereby providing a working approach for large-scale heterogeneous distributed systems and federations.

The architecture of the FELIX test-bed, discussed thoroughly in [2], is depicted in Figure 5. It is composed of various modules that implement the functionalities like, resource orchestration, domain resource management, monitoring, authentication (authN), authorization (authZ), and user access. The Resource Orchestrator ('*RO*)' is responsible for orchestrating the end-to-end network service and resources reservation in the entire infrastructure, as well as delegating end-to-end resource and service provisioning in a technology-agnostic way. An RO usually connects to multiple Resource Managers (**RM**s), which control and manage different kinds of technological resources in a similar way to the Component Managers of SFA [3]. FELIX supports on-demand connectivity between the geographically dispersed federating test-beds, which is realized through the Transit Network RM (**TNRM**) and Stitching Entity RM (**SERM**). They are capable of providing required connection through the transit network domains and manage the physical devices by using frame, packet, or circuit switching technologies; whilst able to support different protocols. The Software Defined Networking RM (**SDNRM**) manages the user traffic environment and the network infrastructure, composed of SDN-enabled devices, by updating the flow tables of the physical devices. The Computing RM (**CRM**) is responsible for setting up and configuring computing resources, e.g., creating new virtual machine instances, configuring network interfaces, etc. Similarly, other essential functionalities are realized dedicated modules such as, the Authentication, Authorization and Accounting (**AAA**) to control user access and the Monitoring System (**MS**) to retrieve, aggregate and store metering information from networking and computing resources.

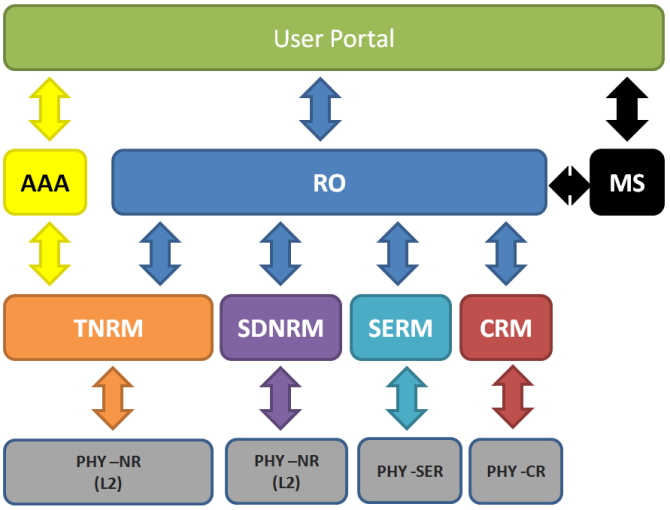

Fig. 1. Architecture of a FELIX test-bed

The **User Portal** is the Expedient [4] based user-agent that offers an intuitive access to the life-cycle management of experiments for experimenters as well as facilitates general management operations for the administrators. To do this, the User Portal communicates with the underlying modules of the FELIX architecture to use a subset of their functionalities to enable experimenters to list the available resources, allocate or provision them, perform operational actions, and release them when no longer needed. On the other hand, the administrators may configure and manage resources, define different types of policies, grant or revoke resource requests and monitor different sets of resources, etc. In addition to User Portal, experimenters can also get access to FELIX test-bed using GENI (www.geni.net) compliant user-agents like OMNI [5] and jFed (jfed.iminds.be).

FELIX uses a combination of recursive and hierarchical configurations for orchestration, request delegation and inter-domain dependency management. The RO entities that are responsible for the synchronization of resources available in particular administrative domains interact with each other via the Master RO (MRO). FELIX has deployed two MROs; one in Europe that interconnects four test-beds in the region, and the other in Japan that links two federating test-beds in the country. An MRO not only performs the inter test-bed resource orchestration but also serves as a trust anchor to facilitate authN and authZ procedures in the federated facilities.

The rest of the paper provides an implementation level detail of the above architectural components and also describes a use-case realized using the federated test-beds of FELIX.

II. Implementation

A. Resource Orchestrator (RO)

The Resource Orchestrator (RO) is a software entity in charge of planning and provisioning the heterogeneous resources of the FELIX Control Framework. It covers different functionalities such as, mediation between the experimenter and the Resource Managers (RM), enforcement of the proper work-flow for the resource reservation of the end-to-end services, maintenance of a high-level and cross-islands topology view and providing aggregated information regarding the status of the physical infrastructure (i.e. servers, switches, devices, etc..) and virtual resources. For these purposes, the RO contacts different RMs through a well-known interface (GENIv3), widely adopted for test-bed federation and therefore implemented in the RO and RMs. Basically, it consists of an XML-RPC with a small number of methods that expect a standardized model, called RSpec, and a subset of options to better define the expected behavior.

The RO is developed as an event-based application composed of a set of logically interconnected modules and libraries. Its architecture is composed of three main layers: the North-bound Interfaces, the Core Components and the Plug-ins, as shown in Figure 2.

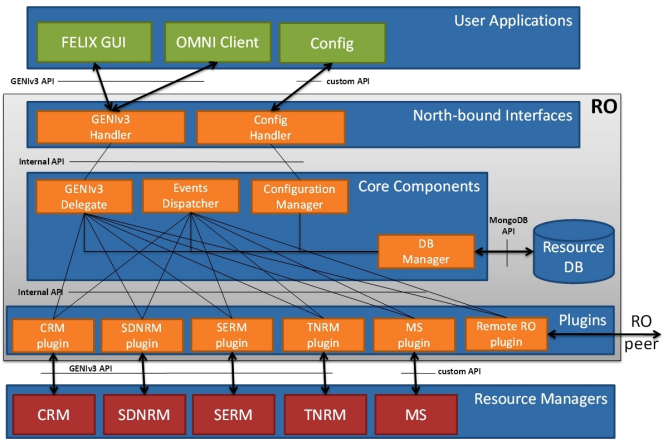

Fig. 2. RO design model

The top layer (*North-bound Interfaces*) contains the components that interact with the GENIv3 interface. Those components provide methods for the management of resources, such as allocation, provisioning and release; and a custom API that allows the configuration of the internal components. Once the experimenter issues a request, the **GENIv3 Handler** delivers it to the *GENIv3 Delegate*, where the required operations are performed. Once completed, the handler formats the response with the appropriate signature. On the other hand, the **Config Handler** manages the configuration messages and invokes the *Configuration Manager* with the incoming parameters.

The middle layer (*Core Components*) contains the modules that perform the main operations, e.g., managing the user requests, discovering the physical topology and generating events for the resources realignment. The **GENIv3 Delegate** executes the GENIv3 methods retrieving data from the database, calling the RM plug-ins and aggregating the results. The **Events Dispatcher** collects the events generated by the other components and then schedules the execution through the dedicated plug-in. The **Configuration Manager** configures the internal components and (just in case) stores the data into the database, e.g. intra-island RMs configuration details, etc. The **DB Manager** provides a wrapper to the MongoDB API and introduces proper filters for the FELIX resources.

The bottom layer (*Plug-ins*) is composed of plug-ins, each of which contact a corresponding RM via a GENIv3 API. The univocal correspondence between plug-ins and RM can be observed in the figure above. The communication to the **MS plug-in** differs from other RMs, as the interfaces and workflows change. In this case, the RO stores information of the infrastructure and slice information, obtained either via user requests through its northbound interface or via calls to the RM through its southbound interfaces. Such information is posteriorly sent to the Monitoring System. In this manner, the Monitoring System is able to update the physical or virtual topology of the underlying resources. Finally, the RO is able to act as a Master Resource Orchestrator (MRO) by using the **Remote RO plug-in**, which interacts with another Remote RO, effectively implement the top-down hierarchical management approach required by the FELIX architecture.

B. Transit Network RM (TNRM)

The Transit Network Resource Manager is the module responsible for the inter-domain networking connectivity. The main function of the TNRM is for network provisioning between SDN slices and Network Services Interface (NSI CSv2 [6]) domains, by acting as a proxy between the RO and NSI CSv2 networks. Specifically, the TN-RM is responsible for: 1) integrating with a different L1/L2 technologies in different network domains, 2) triggering connection operations on southbound interfaces, and 3) communication with its RO, to receive requests and to notify RO about resource status, success or failure events.

The prototype presented here is based on the concepts of GENI architecture and the OFELIA [7] Aggregate Managers and offers similar northbound APIs as other FELIX manager modules, i.e. GENIv3 API, for management communication consistency. It is controlled by Resource Orchestrator (RO), which orchestrates different types of Resource Managers (RM) in the island in a many-to-one relationship. Both eiSoil (github.com/EICT/eiSoil/wiki) and NSI CSv2 were leveraged for this effort. The eiSoil library provides the necessary ”glue” between communication handlers and management logic as well as facilitates common tasks in RM development, which reduces duplication of work. The Java based NSI CSv2 plug-in is implemented to manage NSI CSv2 [8] Service Termination Points (STPs).

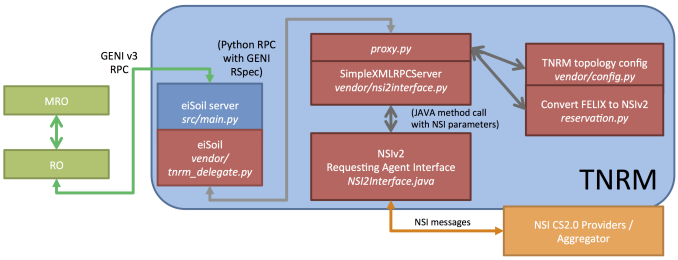

Fig. 3. Diagram of the 5 processes needed to communicate between the client and the NSI domain, via the TNRM.

Semantically, the TNRM maps and translates the GENI v3 state machine to the NSI CSv2 state machine. The five general processes that are executed between the Client and NSI CSv2 domain are shown in Figure 3. In Step 1, the user makes a request for resources through a client that is passed on to the MRO/RO. The resources are described with GENI v3 RSpec XML documents [9]: Advertisement (presents what resources are available), Request (specifies the resources to be used by the client), and Manifest (shows the status of resources). In Step 2, the client obtains credentials from the Clearing House (CH) granting permission for the user request. In Step 3, eiSoil calls *tnrm_delegate.py* to implement the GENIv3DelegateBase. The *tnrm_delegate.py* in turn calls *proxy.py*, which is the front end of the SimpleXMLRPCServer to communicate with the NSI CSv2 NRM or AG. This is done by calling *nsi2interface.py* to invoke a Jython call to NSI2Interface.java. This Jython step is necessary in order to translate the Python calls of eiSoil to the Java calls in the NSI CSv2 agents. Lastly, in Step 5, *NSI2Interface.java* calls the NSI CSv2 Requester to connect to NSI CSv2 provider (AG, NRM, etc.) by NSI CSv2 protocol using a CXF web service. It is important to note that Step 4 also requires network topology information. This is obtained by having *proxy.py* call a *config.py* script that reads a configuration file (*config.xml*) defined at a TN terminal point, which contains node, interface, and NSI CSv2 STP information. It is important to note that in order to communicate between the NSI CSv2 Java calls and the eiSoil Python calls, a third language Jython was implemented. This added level of complexity can be extended to make the TNRM more flexible, giving it the ability to meet the requirements of any network domain.

C. Stitching Entity RM (SERM)

Stitching Entity Resource Manager interconnects the external connection points with the local SDN island by the switching rules configuration on the switching device (Stitching Entity). In order to make its architecture design consistent with other FELIX modules, the common part (Delegate, Parsers, Formatters) is based on the RO module implementation. Moreover it offers northbound APIs similar to other FELIX manager modules, i.e. the GENIv3. The SERM allows an experimenter to request, update and delete stitching resources through switching rules. This module can also act as a proxy between an RO and the stitching device. It receives requests from the RO, checks the availability of switching hardware and performs requested configuration. The SERM is also responsible for triggering the tear-down of expired reservations.

Figure 4 presents the SERM components and their interconnections. **SERM configuration file** is a file that specifies

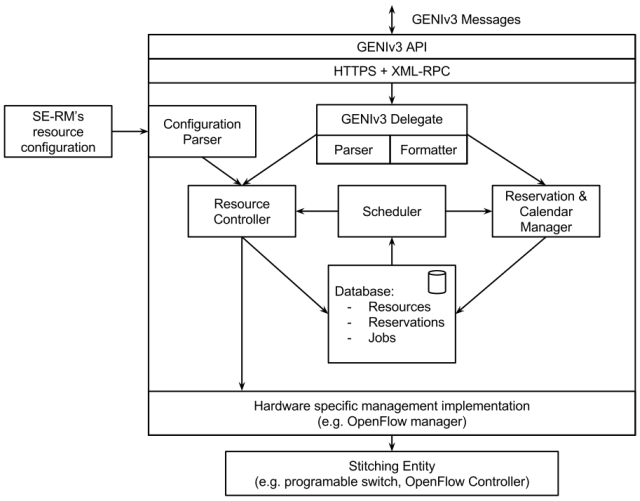

Fig. 4. Stitching Entity Resource Manager components

which ports on the switching hardware take part in the FELIX test-bed (NSI, STPs and static links). Some additional parameters can also be specified like VLAN tagging options or FELIX-specified identifiers. The **Configuration parser** extracts the initial configuration, then checks if change could interrupt the ongoing reservations and transforms all data according to data template stored in the SERM Database. The **GENIv3 Delegate (with Parser and Formatter)**: allows aggregates advertise and allocate resources to slices in the form of slivers. This component is also responsible for parsing. The **Resource Controller** gets the requests for deploying a connection between the ports (switching rule), checks their feasibility (e.g., through checking available port and/or VLAN in configured range) and triggers them by translating real ports values through the hardware-specific management implementation. The **Reservation & Calendar Manager** is responsible for handling RSpec request in reservation context and stores all needed information in dedicated data model. Additionally it also works as calendar for managed resources. Moreover this component takes part in releasing resources process. The **Database** stores the status of the stitching resources like ports and associated VLANs. It also stores the on-going reservation with the deployment status and keeps scheduled timeout events. Finally, it stores information about resources to be released when reservation time expires. The **Hardware specific management implementation** is a pluggable component that implements management features of a specific hardware and management interface, e.g. an SSH client that invokes the CLI commands on the remote console or the OpenFlow client app.

D. Compute RM (CRM)

The Computing Resource Manager module allows the experimenter to provision and manage Virtual Machines (VMs) on a number of physical servers. The CRM is based on the Virtualisation Technology Aggregate Manager (VTAM) of the OFELIA Control Framework (fp7-ofelia.github.io/ocf). As with other Resource Managers in FELIX, the management and provision through CRM is possible thanks to a well-known interface (GENIv3) and a custom data model (RSpec) filled up with the requested data. The GENIv3 interface has been widely

adopted for test-bed federation. Figure 5 shows a detailed schema of the CRM architecture.

An experimenter's request arrives from one of the XML-RPC APIs (either **GENIv3 API** or **SFA (GENIv2) API** [10]) or from the OCF's Graphical User Interface (GUI), which communicates with the custom **OFELIA API**. All APIs are located in the top layer (*Northbound APIs*). After the experimenter's credentials are properly verified in the upper layers, its request traverses the core components located in the layers below.

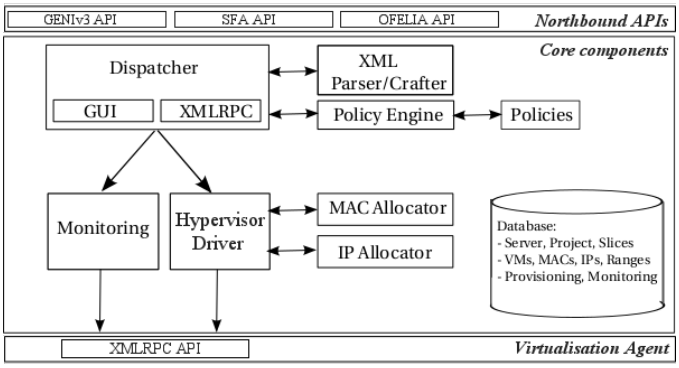

Fig. 5. Components of the FELIX C-RM

The middle layer (*Core components*) addresses the request by forwarding it to the appropriate components. Specifically, the **Dispatcher** component intercepts and forwards the request to one or another component, as needed. Depending on the entry point of the request, the Dispatcher component processes the request sent through the *GUI* or the *XMLRPC* subcomponents. First, the request is parsed and interpreted in the **XML parser**. Then, the requirements (e.g. quantity of memory, template name) are evaluated against the specific policies of the domain, previously defined through the **Policy Engine**. If the request complies with the local policies, the request is processed and new data is computed (such as the MAC and IP used for provisioning) through the **MAC Allocator** and **IP Allocator** components. After this, the new information is included in a properly formatted request and forwarded to the XML-RPC API of the *Agent*, which is located in every managed server. The information of the machine recently reserved or provisioned is persisted in the internal database. Finally, and in compliance with SFA [3], once the VM is provisioned and running, the experimenter's public key is inserted to grant access to the VM.

The bottom layer (*Virtualisation Agent*) hosts the **Agent** component and all its related components and utils. The physical infrastructure may consist of any of two different kinds of hypervisors: XEN, as per the original VTAM module, or KVM, which was extended during the project. In both cases, the agent receives an XML-formatted request, previously sent from the *Core layer*, through its northbound API. The Agent interprets the data from the request and communicates with the custom drivers of the given Virtualisation Hypervisor in order to create the VM. Once the process is finished and the VM is created, the Agent sends a notification back to the other subcomponents from the *Core layer* that are involved on the process (i.e. the Expedient and the administrative panel of the CRM's GUI).

E. SDN RM (SDNRM)

The Software-Defined Networking Resource Manager allows the experimenter to define a set of rules to forward packets through the OpenFlow switches, given a set of matches and actions. For this, the SDNRM communicates with an external specific-purpose controller, named *FlowVisor*, that slices the traffic between different experiments. The SDNRM is based on the OpenFlow Aggregate Manager (OFAM) of the OFELIA Control Framework (fp7-ofelia.github.io/ocf). As with CRM, the SDNRM uses the GENIv3 API for the definition of flows and RSpec as custom data model. Figure 6 shows a detailed schema of the SDNRM's architecture.

The experimenter's request arrives from a pertinent XML-RPC API (either **GENIv3 API** or **SFA (GENIv2) API** [10]) or from the User Portal that communicates with the custom **OFELIA API**. All APIs are located in the top layer (*Northbound APIs*). After the experimenter's credentials are properly verified in the upper layers, its request traverses the core components located in the layers below.

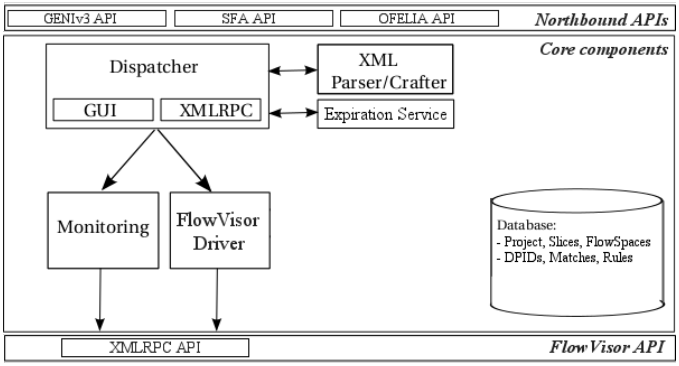

Fig. 6. Components of the FELIX SDNRM

The middle layer (*Core components*) interprets the request and operates on it as needed. Depending on the entry point of the request, the Dispatcher component processes the request sent through the *GUI* or the *XMLRPC* subcomponents. When the request is received through the *GUI*, the set of matching conditions (where VLANs are typically used) may need further processing to define the specific VLANs from a given number requested by the experimenter. This is done internally in this layer by a subcomponent called *VLAN Manager*. This subcomponent is able to contact the different networks involved to agree on a common subset of VLANs to be used. Such operation could be useful in case no SERM was used for stitching. The request, in any case, is parsed and interpreted in the **XML parser**. Then, the request is set an expiration date and stored in the internal database. The **Expiration Service** performs a periodic review of the expiration date of the existing requested flowspaces and deletes it if necessary. Before the request is sent to the *FlowVisor*, the SDNRM performs an internal verification of the matching conditions requested by the experimenter, such as whether the VLANs requested are available for new experiments. In this regard, the group's section of the request is analyzed to avoid collisions with set of rules (*flowspaces*) already granted. After this process, the request is ready to be transmitted to the FlowVisor through the **FlowVisor Driver**.

The bottom layer (*FlowVisor API*) hosts the **FlowVisor**

Proxy. This proxy interacts directly against the XML-RPC interface of the FlowVisor controller. Once the controller finishes its internal operation, it sends a response back to the intermediate layer, which subsequently notifies other subcomponents involved in the process similar to CRM.

F. Monitoring System (MS)

The Monitoring System (MS) is the module that monitors slice(s) over the FELIX test-bed. The MS retrieves the monitoring data of such resources with the help of the local RO and provides it to the experimenters and island administrators. The MS module is composed of three components as depicted in Figure 7. i) The **Topology API** manages the resource and topology information with a proprietary API. This API accepts the topology and resource information subject to monitoring from the RO. It also provides the topology information to external entities such as an experimenter's client or monitoring GUI. ii) The **Monitoring Data Collector** (MDC) gathers the monitoring data from the physical resources using third-party monitoring tools, e.g., perfSONAR (www.perfsonar.net) and Zabbix (www.zabbix.com), which retrieve measurements directly from the physical hardware. The MDC gathers such data through the monitoring tools' APIs and inserts it to the Monitoring database. MS adopts a hierarchical structure similar to the RO. Another feature of MDC is to forward data from an MS to the Master MS (MMS), upper layer MS, or to exchange information between different MMS. iii) **Monitoring API** manages the monitoring data through a proprietary API. This API accepts incoming monitoring data and stores it into the Monitoring DB. The MDC stores the monitoring data in MS Database through this interface. In addition, this API provides interface to the external entities for requesting the monitoring data.

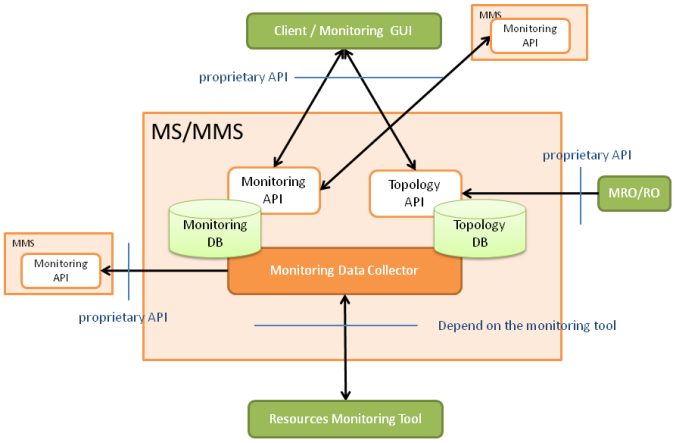

Fig. 7. Components of the MS

G. Authentication and Authorization

C-BAS (www.eict.de/c-bas) [11] employs X.509 certificates and SFA styled credentials to realize AAA services in FELIX. The implementation of C-BAS was based on eiSoil (github.com/EICT/eiSoil/wiki) to exploit its plugin capabilities that enable importing the functionality from one plugin module to another. C-BAS implements its functions through plug-ins as shown in the UML package diagram (Figure 8). The **fedrpc** plug-in module builds the service access interface of CBAS to receive RPC calls of API methods. The **ofed** maps the access API calls onto the internal clearinghouse methods and handles authN/authZ through the use of **geniutils** plugin offered by GENI. The **fedtools** comprised helper functions that include, conformation checks of arguments passed in the RPC method calls, verification of credentials and privileges, as well as, management of supported member roles and their default set of privileges.

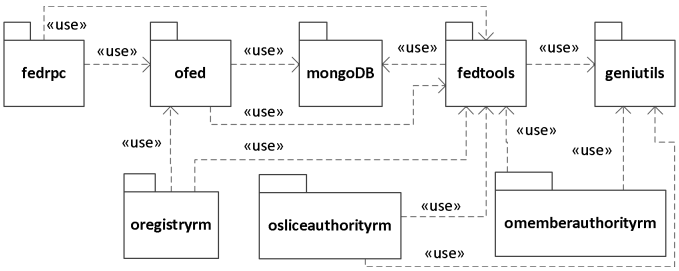

Fig. 8. UML package diagram of C-BAS

The **omemberauthorityrm** implements the Member Authority (MA) services to offer management of member information and SSH keys. It maintains a database of registered members' credentials, certificates, and SSH keys. It also registers new members and issues certificates and credentials based on their assigned roles. Moreover, it is consulted to update, lookup, and delete the member information. In addition, it maintains a Certificate Revocation List which contains invalidated member certificates. This list is periodically updated and disseminated to other system entities involved in authN/authZ. The **osliceauthorityrm** plug-in module realizes the Slice Authority (SA) services that include management of projects, slices, slivers and user credentials for projects/slices. The slice credential issued by SA asserts a user's membership and privileges for a slice and serves as means to authorize GAPI [10] calls to RO/RMs. SA also facilitates the creation, update, lookup, and removal of projects, slices and their memberships.

The **oregistryrm** implements Service Registry (SR) which holds pointers for all C-BAS services (e.g., SA, MA) and serves as a primary contact point for the test-bed. The information offered by SR is statically configured in C-BAS. The **mongodb** realizes persistence for C-BAS by implementing a lightweight layer over noSQL database to help execute database queries like create, delete, update, and lookup of entries in a collection.

C-BAS offers its services to the User Portal, TNRM, and other user-agents like OMNI and jFed. The use of certificates in C-BAS greatly simplifies the authN/authZ mechanisms when it comes to federating FELIX test-beds, because all that is required is to establish mutual trust among the test-beds by marking their root certificates as trusted. This enables the validation of certificates and credentials issued by one test-bed at all other federating test-beds within FELIX.

III. Use Case

High Quality Media Transmission over long-distance networks is one of the use cases proposed by FELIX [2] and assesses the FELIX Control Framework for provisioning SDN resources for experiments. The provision process involves dynamically creating a virtual slice over the federated infrastructure of European and Japanese test-beds. The slice

allows to demonstrate the capabilities of the involved test-beds and judge the test-bed fitness for long-distance, high-quality media streaming. This is because the demands and requirements for streaming technologies make them highly sensitive testing engines to detect transmission problems. The technology developed for this use case to automatically adjust connection paths and parameters can be leveraged to detect network failures and problems. The multimedia streaming tool used in these experiments offers a number of quality measurement capabilities for evaluation purposes. In this use case, two studies were performed: i) examining long distance network capabilities and evaluating user experience (QoE), and ii) examining a new intelligent network application to control high quality media streaming.

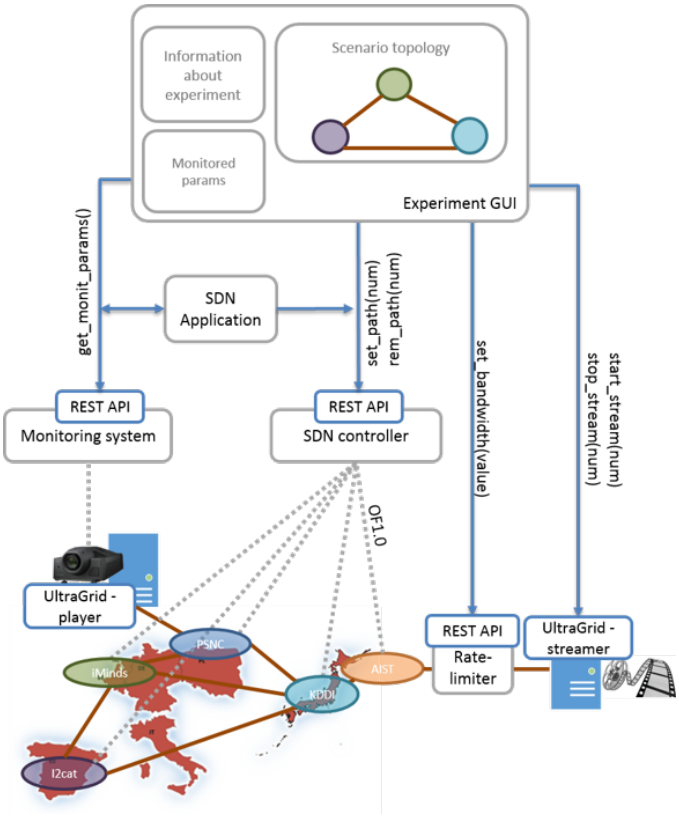

Fig. 9. An overview of experiment and provisioned resources.

Figure 9 depicts some of the experiment configurations and the employed infrastructure. The procedure to perform the experiment is as follows. Two physical machines (media source and consumer) are reserved on two different federated test-beds and UltraGrid streaming software (www.ultragrid.cz) is installed on both machines. Then a network connectivity is dynamically established between the two machines. Instead of following a direct path, traffic is deliberately routed via other test-beds as shown in Figure 9 for evaluation purposes. Finally, using the UltraGrid a high quality media transmission is started at one end and visualized at the other end. During the tests, the experimenter may monitor various performance metrics of the network connections, e.g., bandwidth, round trip time, correctly decoded video frame per second and lost video frames due to reordering, jitter, excessive delay, etc.

These experiments provide information on the impact of various network parameters on the transmission quality. Specifically, i) it checks the boundary condition of the network capacity needed for transmission of the high quality media (based on user QoE) and thus provides information about the minimum bandwidth required for media delivery with satisfying quality, ii) it provides real time monitoring of the media traffic to observe possible stream degradation due to traffic congestion on the paths, and iii) it examines the behavior of the smart network application for automatic network configuration and adjustments during the degradation of parameters.

IV. Conclusions and Future Work

Having completed the implementation of the FELIX architecture and the start of one use case to validate and test the architecture, the next step is to perform the other five use cases to further validate and test the resulting implementation. In parallel, various features of the software stack will be refined as necessary for each use case. The completed implementation will facilitate the federation and integration of different network and computing resources that are distributed in multiple domains across different continents. Experimental users of the testbed will be able to request, provision, manage and monitor various resources to create a slice in a heterogeneous, geographically distributed environment. This ability to dynamically create network slices using federated infrastructures is expected to encourage closer and deeper collaboration in FI research and development, especially between EU and Japanese communities.

Acknowledgment

This work was conducted within the framework of the EU FELIX projects, which are partially funded by the Commission of the European Union.

References

[1] G. Carrozzo, et al., "Large-scale SDN experiments in federated environments," in *Proc. SACONET WOSDN*, March 2014.

[2] C. Fernandez, et al., "Large-scale SDN experiments in federated environments," *IJPEDS Volume 30, Issue 3*, April 2015.

[3] L. Peterson, et al., "Slice-based Federation Architecture (SFA)," July 2014.

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

[5] "The Omni client," http://trac.gpolab.bbn.com/gcf/wiki/Omni, June 2015.

[6] T. Kudoh, et al., "Network services interface: An interface for requesting dynamic inter-datacenter networks," *Optical Fiber Communication Conference (OFC)*, Mar. 2013.

[7] M. Sune, et al., "Design and implementation of the OFELIA FP7 facility: The European OpenFlow testbed," *Computer Networks*, 2014.

[8] G. Roberts, et al., "Network service framework v2.0," Jun. 2014.

[9] "GENI v3 RSpec Schema," http://groups.geni.net/geni/wiki/RSpecSchema3.

[10] "The GENI Aggregate Manager API," http://groups.geni.net/geni/wiki/GAPI_AM_API, 2013.

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

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.