



## 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 D3.4 End User Tools and API

Version 1.0

Due date: 31/11/2014  
Submission date: 30/01/2015  
Deliverable leader: i2CAT  
Author list: Carlos Bermudo (i2CAT), Carolina Fernandez (i2CAT), Umar Toseef (EICT), Kostas Pentikousis (EICT), Bart Puype (iMinds), Takatoshi Ikeda (KDDI)

#### Dissemination level

- |                                     |   |
|-------------------------------------|---|
| <input checked="" type="checkbox"/> | PU: Public  |
| <input type="checkbox"/>            | PP: Restricted to other programme participants (including the Commission Services)        |
| <input type="checkbox"/>            | RE: Restricted to a group specified by the consortium (including the Commission Services) |
| <input type="checkbox"/>            | CO: Confidential, only for members of the consortium (including the Commission Services)  |

**<THIS PAGE IS INTENTIONALLY LEFT BLANK>**

# Table of Contents

<b>Abstract</b>	<b>6</b>
<b>Excecutive Summary</b>	<b>7</b>
<b>1 Introduction</b>	<b>8</b>
<b>2 Definitions</b>	<b>9</b>
2.1 Abbreviations . . . . .	9
2.2 Definitions . . . . .	9
<b>3 Implementation Details</b>	<b>10</b>
3.1 Expedient . . . . .	10
3.1.1 Design . . . . .	10
3.1.2 Workflows . . . . .	14
3.1.3 Future Work . . . . .	17
3.2 AAA . . . . .	18
3.2.1 Design . . . . .	21
3.2.2 Workflows . . . . .	25
3.2.3 Performance Evaluation . . . . .	27
3.2.4 Future Work . . . . .	30
3.3 Public Monitoring . . . . .	30
3.3.1 Design . . . . .	31
3.3.2 Workflows . . . . .	32
3.3.3 Future Work . . . . .	33
<b>4 Deployment</b>	<b>35</b>
4.1 Expedient . . . . .	35
4.1.1 Requirements and Dependencies . . . . .	35
4.1.2 Configuration and Installation . . . . .	35
4.1.3 Operation . . . . .	37
4.2 AAA . . . . .	39
4.2.1 Requirements and Dependencies . . . . .	39
4.2.2 Configuration and Installation . . . . .	39
4.2.3 Operation . . . . .	39
4.3 Public Monitoring . . . . .	40
4.3.1 Requirements and Dependencies . . . . .	40
4.3.2 Configuration and Installation . . . . .	40
4.3.3 Operation . . . . .	40
<b>5 Conclusions and Summary</b>	<b>42</b>
<b>References</b>	<b>43</b>
<b>Appendix I</b>	<b>44</b>
A C-BAS API methods . . . . .	44
A.1 Slice Authority (SA) API methods . . . . .	44
A.2 Member Authority (MA) API methods . . . . .	45
B API method of FAPI Plugin for Expedient . . . . .	46

## List of Figures

Figure 3.1	Expedient design . . . . .	11
Figure 3.2	Expedient Topology screen . . . . .	12
Figure 3.3	Expedient Resource status window . . . . .	13
Figure 3.4	Certificate-based user authentication with Expedient and C-BAS . . . . .	14
Figure 3.5	Expedient's user login webpage . . . . .	15
Figure 3.6	Expedient's interface for user certificate and SSH keys management . . . . .	15
Figure 3.7	Expedient workflow . . . . .	16
Figure 3.8	User registration process . . . . .	17
Figure 3.9	Certificate-based identity verification and authentication . . . . .	19
Figure 3.10	Member role hierarchy . . . . .	20
Figure 3.11	Delegation of privileges. (a) delegated credentials, (b) speaks-for credentials, (c) speaks-as user . . . . .	20
Figure 3.12	C-BAS service interfaces . . . . .	22
Figure 3.13	UML package diagram of C-BAS . . . . .	24
Figure 3.14	Project credentials lookup and slice creation processes . . . . .	26
Figure 3.15	Expedient's interaction with C-BAS and RM to request resource allocation . . . . .	27
Figure 3.16	Boxplots of request processing times for user registration at C-BAS. . . . .	28
Figure 3.17	Boxplots of request processing times for project credentials lookup and slice creation. . . . .	29
Figure 3.18	Boxplots of request processing times for user authentication through certificate verification and user credentials lookup. . . . .	29
Figure 3.19	Boxplots of request processing times for project and slice credentials lookup at C-BAS. . . . .	30
Figure 3.20	Public monitoring application interactions . . . . .	31
Figure 3.21	Requesting island status page . . . . .	32
Figure 3.22	Updating SDN resources . . . . .	33
Figure 3.23	Updating computing resources . . . . .	34
Figure 4.1	Example public monitoring web page (PSNC) . . . . .	41

List of Tables

Table 3.1	Member roles and associated privileges . . . . .	25
Table 4.1	Configurable paramters . . . . .	36
Table 4.2	Island manager account parameters . . . . .	36
Table 4.3	Database configuration parameters . . . . .	36

## Abstract

This document details the end-user tools and APIs used for managing the infrastructures and setting up and running the experiments in the FELIX Federated Framework. An overview of the design and implementation of the new features for the OFELIA Expedient Control Framework, the AAA module and the public monitoring tools is given; as well as explaining their key internal and inter-communication workflows.

## Excecutive Summary

Deliverable D3.4 purpose is to explain the main design, implementation and deployment concepts and decisions taken to provide the FELIX project with end user tools and APIs.

The document is structured in two main parts: the implementation details and the deployment. After a brief introduction and a list of acronyms and short definitions of the concepts used in the document, the Implementation Details section presents the chosen tools to provide the FELIX users access to all its functionalities and how to manage the credentials and permissions to perform them. For those tools, a high level view of their design and basic workflows is provided.

After that, the other main section of the document, Deployment, provides instructions on how to install, configure and manage those tools. Main requirements are listed prior to the step-by-step instructions to install and the main variables to configure them. A short instruction manual on how to use the tools helps the users on their first steps.

The document finishes with the Conclusions and Summary and a technical Appendix with the methods of the different APIs used in the previously described tools.

This document is addressed to software architects, software and network engineers, software developers implementing specific features of the end user tools and APIs, as well as to the system administrators.

# 1 Introduction

The top level layer of the FELIX framework comprises the GUI based user-agent, a set of CLIs and few internal modules.

The user-agent offers testbed access to the users, both administrators and experimenters. Through the GUI, the administrators can view information about the infrastructure and the domain, as well as get control over the resources. For example, the AMs from OFELIA have their own webpage for administration.

The main user tools presented in this document are the Expedient (an adaptation of the GUI-based user-agent employed in the OFELIA project), the AAA module named as C-BAS and the public monitoring tool.

To experimenters, Expedient offers functionalities to request the creation of a project (a container for slices or experiments) and slices and add RMs to them. The users also have access to a history of their actions and system messages. They can manage user privileges over their projects and update/modify their own personal information, SSH keys, certificates etc.

To administrators, Expedient is the tool to manage the infrastructure. They can add indirectly the RMs of the other domains through the RO, check logs of the infrastructure and all the experiments' status plus the information about all the projects, slices and users of that domain. A user's registration to the system as well as all his requests to create projects must get approval from the administrator of the island. In addition, it's the administrator who decides to which islands to federate.

The C-BAS module helps keep consistency in the information and manages the permissions of the users, projects, slices, etc. It has a distributed architecture, i.e., there is a C-BAS module per island but all C-BAS modules exchange important information like, certificate revocation lists. C-BAS module interacts with Expedient in several ways as it serves as its clearinghouse. Therefore operations like, user registration, membership renewal, creation of slice or projects, credentials generation or retrieval etc. are performed through C-BAS at the back-end while GUI from Expedient runs at the front-end.

The public monitoring tool provides live status and general overview of the available infrastructure also useful for both administrators and experimenters. The former can get noticed when something is wrong in the infrastructure (server down, unreachable switch, etc.) and the latter can know which resources are available for the experiment.



## 2 Definitions

Throughout this document we use specific notation and acronyms that are explained here. Please refer to this guide to identify the concept or for a more detailed explanation.

### 2.1 Abbreviations

- **AAA:** Authorization, Authentication and Accounting
- **AM:** Aggregate Manager
- **CLI:** Command Line Interface
- **GUI:** Graphical User Interface.
- **LDAP:** Lightweight Directory Access Protocol
- **OFELIA:** OpenFlow in Europe Linking Infrastructure and Applications
- **OCSP:** Online Certificate Status Protocol
- **OFVER:** OFELIA VERsioning system.
- **RM:** Resource Manager
- **SFA:** Slice Federation Architecture
- **VM:** Virtual Machine.

### 2.2 Definitions

- **Authorities** are the services that manage assertions about members and their permissions with respect to aggregate resources.
- **Island:** Physical domain under particular management. It provides infrastructure and resources to the end user.
- **Members:** In this document experimenters using the facility are called *members*.
- **Projects:** Groups of slices and members for a particular administrative purpose.
- **OFVER:** Versioning system that consists of a number of core scripts to manage the install and update processes, and allows extension through custom scripts.
- **Slice:** For practical purposes it makes sense to group resources at RMs as so-called *slices*. It is set of slivers spanning a set of network components, plus an associated set of users that are allowed to access those slivers for the purpose of running an experiment on the substrate.
- **Sliver:** A portion of a resource that has been granted to a member
- **User-agent:** Members typically employ *user-agents*, i.e. software tools which interface with RMs and authorities on behalf of members.

### 3 Implementation Details

This section provides the implementation related information about the end user tools (GUI and public monitoring) and the APIs. It will explain the developed/adapted/improved modules, the design decisions taken, how they work and what future functionalities are missed/desired.

There are three main modules used as end user tools and APIs in the FELIX project. The first module is the end user tool from the OFELIA project web-based GUI, Expedient. It provides very similar functionalities and workflows as required by FELIX.

The second module is the AAA (named as C-BAS), which provides a set of APIs to interact with Expedient to allow access to the resources only to authorised users. In addition to Expedient, C-BAS is also compatible with a 3rd party command line tool OMNI [1] which is shipped with GENI Control Framework (GCF) software package. OMNI supports GAPI to communicate with RMs for resource reservation and supports FAPI with C-BAS to create slice and access user or slice credentials etc.

The third one is the public monitoring tool, which provides an overview of the available infrastructure and is planned to be offered in the FELIX website to help users to have a live view of the status of the components (e.g. switches, servers, etc.) and its connectivity.

The following sections provide the implementation details of Expedient, C-BAS and public monitoring.

#### 3.1 Expedient

In order to follow T3.7 guidance about reusing existing tools as much as possible and taking into account T2.8 conclusions provided in D2.2[2], Expedient has been adopted from OFELIA project[3] to offer users with an interface for exposing and managing different testbeds resources. In FELIX, Expedient also provides management interface for C-RM and SDN-RM (based on the VTAM and OFAM in the OFELIA project). Its webUI offers specific pages for management, experiment setup, etc. Its appearance is easily customizable to change its colours and logos to adapt to each testbed/project "brand" image.

Expedient follows a pluggable architecture as required by T3.7, to support resources available at SE-RM and TN-RM as well as support components such as the RO.

Expedient acts as intermediary between the users and the ROs providing information about the available resources for experimentation and its topology and transforming users petitions into requests in the proper components APIs.

##### 3.1.1 Design

Expedient has two main components from the design view point; (i) the GUI or front-end and, (ii) the back-end including the plugin system.

The GUI allows the interaction of the users with the testbed. It provides functionalities for both experimenters and administrators of the testbed and passes their requests to the back-end and presents the returned responses.

The back-end processes the requests coming from the GUI and sends them to the corresponding relevant components. The back-end also supports a plugin system to extend the Expedient functionalities and hence satisfy the requirements of T3.7. This pluggable architecture allows adding new resources and/or components. In this case, a plugin to communicate with the RO has been developed.

Expedient also interacts with the AAA (i.e., C-BAS) module to validate the users' permissions to perform any action. Figure 3.1 shows the Expedient components and their interactions with other FELIX modules.

##### 3.1.1.1 GUI

Expedient GUI has 2 main different parts, from the user point of view: the administration and the experimentation part. They are properly described in the following subsections.

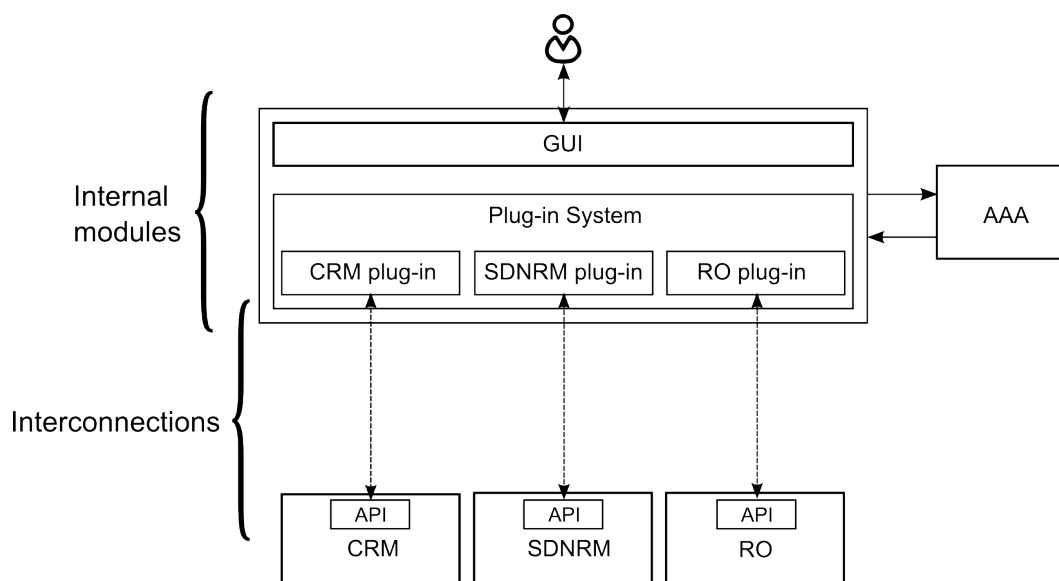


Figure 3.1: Expedient design

### Administration

Expedient allows the testbed managers adding AMs (that is RMs) for the different resources of the different testbeds.

Expedient also facilitates the management of users and permissions. New users can be registered through Expedient and their permissions can be modified. When an Expedient user needs permission to do something, and he does not have that permission, he needs to request that permission from someone who can give it to him. For example, a user can make a request to a project owner (another user) to create a slice in his project.

Administration part interacts with AAA module in order to control who can access the resources.

### Experimentation

Expedient provides a useful graphical interface to the users for managing their experiments. Through Expedient, a user can create a project (a container for experiments) by sending a request to the administrator. Once the project is created, the creator can add other users to the project so that they can also perform experiments in that project.

The user can also add aggregate managers or RMs to the project, that is, the resources that will be available for the experiments.

The experiments are created as slices. After creating a slice, a user can select from AMs provided by that project for use in experiment. Once included, the resources of these AMs are graphically displayed in Expedient along with the links between them.

### Monitoring

A new GUI section for the Expedient has been created in FELIX project which provides information about the status of the components and the slice topology.

### Topology

The topology screen that appears in the Slice section inside a Project has been modified for the Monitoring section in order to show two parallel topologies (see Figure 3.2). The bottom topology window shows the physical topology of the available resources. It depicts all the available resources and their connectivity to the resource

managers of that slice. The resources belonging to the same testbed are painted in the same colour. For example Figure 3.2, represents 2 different islands (green and orange colours) linked through TN-RMs (in red) to the "Transport Network" represented by the pink ellipse.

The upper topology window shows the resources added to the slice, that is, the ones that the user will use in his experiment. The resources shown in this window are placed exactly above their equivalent resource in the bottom window and mapped with a dotted line to the resources in the bottom window to ease the identification. Furthermore, if a resource is moved in any of both windows, its reflection will move the same way to keep the same position. In this case, the slice topology shows only switches and servers, being the SE-RM, TN-RM and transport network represented by a link as they are resources transparent for the experiment.

Apart from the addition of a new window, the screen is also improved by providing more clear information for each node when moving the mouse cursor over the icons. A table with the status of each port of the switch has been added to the displayed information.

### Slice slice1

#### Topology



Figure 3.2: Expedient Topology screen

#### Resource details

The new "Resource details" window is the front-end of the monitoring system, showing statistics of a selected resource during a defined period of time. The window allows having a graphical view of the status or the statistics of a selected switch. The user can select the resource, define a window time (date and hour) and will get a graphical representation of the status (up or down) or incoming or outgoing data rate of the selected switch. The window also allows zooming on the graph or visualizing a table with the numeric data from the graph.

For example, Figure 3.3 shows both the input and output traffic measured in a switch in the time period from 2015/01/01 to 2015/01/31. Once the data is retrieved, the time periods can be modified through the small graph on the right.

#### 3.1.1.2 Back-end

## Resource details

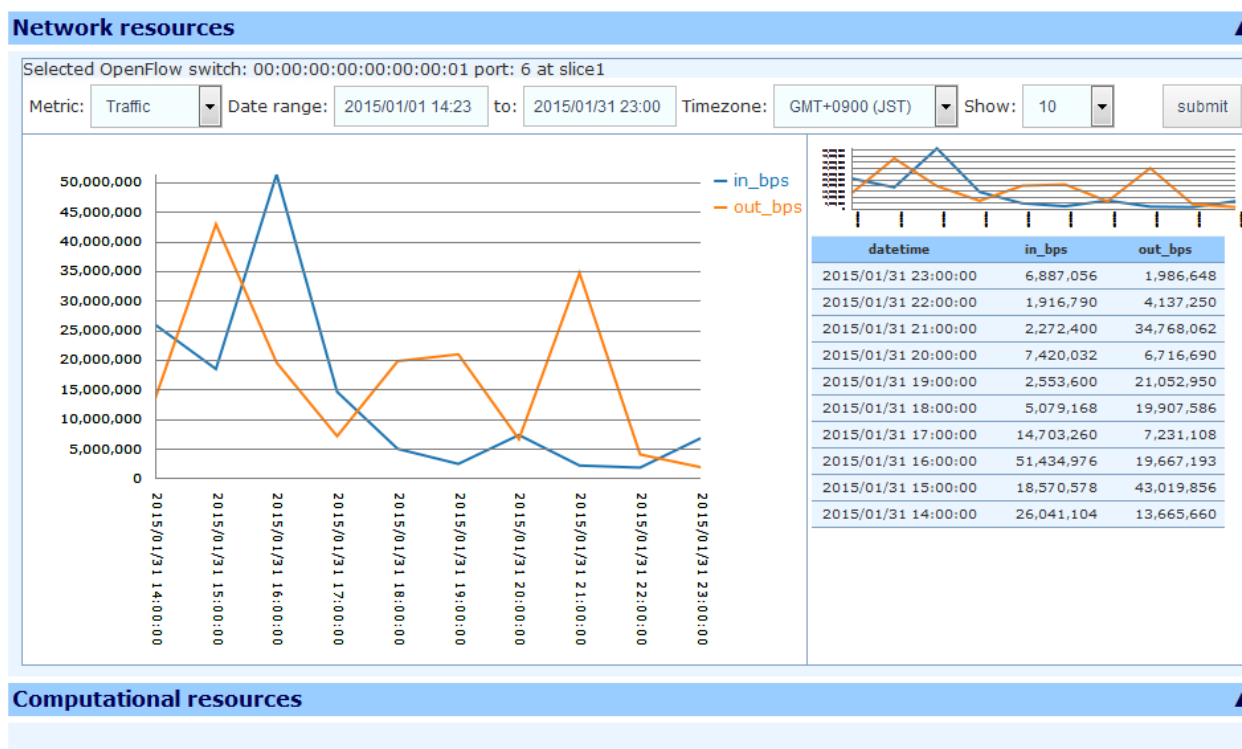


Figure 3.3: Expedient Resource status window

### RO Plugin

As discussed above, OFELIA Expedient has a plugin system which allows adding new resources in the form of Aggregate Managers. When a plugin is added, the slice definition screen includes the management part for the new resource in order to add it to the experiment.

Originally, Expedient offered two kind of plugins, one for the VT AM (Virtualization Aggregate Manager) and another for the OF AM (OpenFlow Aggregate Manager), the equivalent to C-RM and SDN-RM, respectively. In order to manage the other FELIX resources, a plugin to manage RO through Expedient has been developed.

### Certificate-based authentication

Expedient comes with the traditional password based login mechanism which is not encouraged by C-BAS. Instead, C-BAS promotes a more secure certificate and private key based authentication. The private key associated with the certificate consists of randomized text having hard to guess relationship with the public key embedded in the certificate. Its unique cryptographic properties eliminate a number of vulnerabilities inherit to password based authentication mechanism. This work realized certificate-based authentication for Expedient through an extension. Figure 3.4 illustrates the process of user login onto Expedient using the certificate and private key. At login webpage that belongs to Expedient front-end, users are prompted for certificate and private key. Although private key is required in login process to verify that the user who is presenting the certificate is its rightful owner but its transmission all the way to back-end server is not desirable. This issue is resolved by exploiting the unique token that comes embedded in login page to prevent Cross-Site Request Forgery (CREF) attacks. CSRF is an attack that lures the victim to submit a malicious request to website to which victim is currently authenticated. The purpose of such an attack is to perform actions on victim's behalf through inheriting his identity and privileges. To avoid this CSRF tokens are used that helps back-end server distinguish between forged and legitimate requests.

In conjunction with certificate based login, CSRF token is treated as a challenge token from back-end and must be digitally signed by the user's private key. This is performed by a Java script in login page so that private key never leaves user's machine. This digitally signed token along with the certificate is transmitted to the back-end server that verifies the digital signature using the public key embedded in the certificate. If verified it implies user's possession of private key associated with the presented certificate. Next, it is checked if certificate is valid, unrevoked and issued by clearinghouse of C-BAS or one of its federation. This is performed by sending the user certificate to MA for verification. If passed, the user authentication completes otherwise an error message is shown to the user. In final step, the user's system credentials are fetched from C-BAS and cached at Expedient. For this purpose, Expedient performs member lookup at C-BAS by providing its own credentials for authorization. As a result, user credentials are retrieved and used in communication with C-BAS as shown in Figure 3.14 and Figure 3.15.

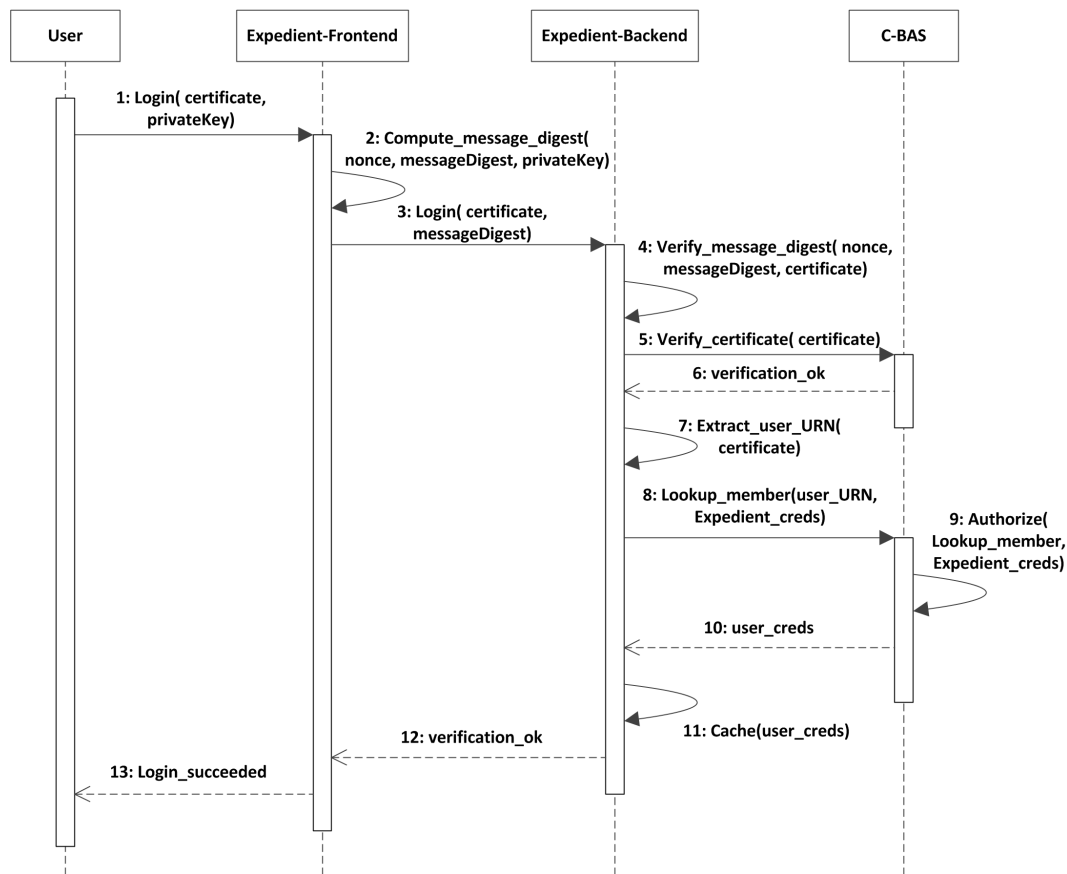


Figure 3.4: Certificate-based user authentication with Expedient and C-BAS

Figure 3.5 shows Expedient's webpage for certificate-based user login while Figure 3.6 shows the user account area designed for certificate and SSH key management.

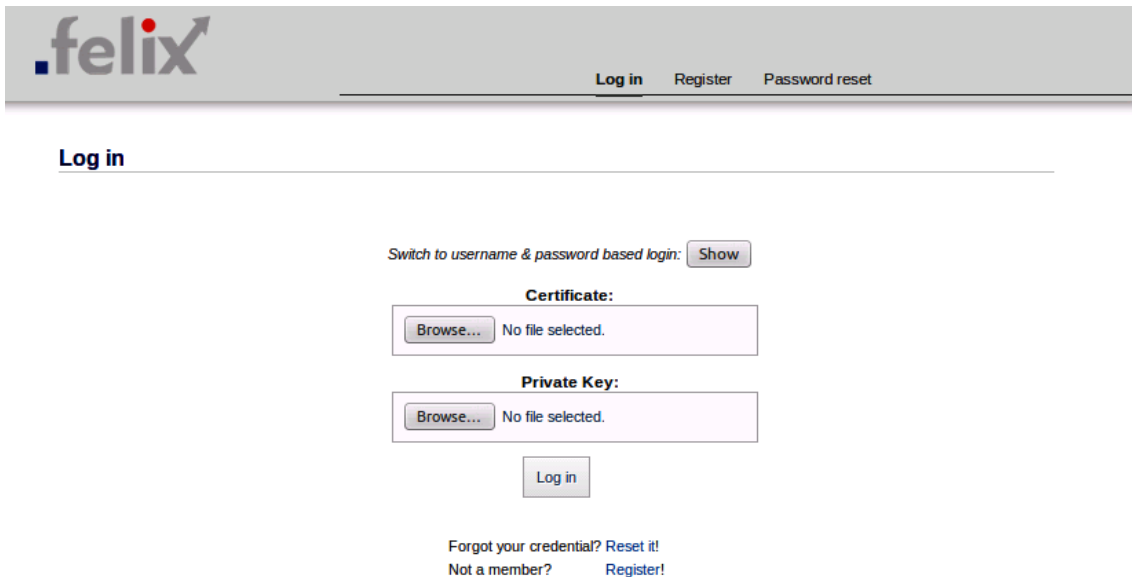
### API for C-BAS

In order to facilitate Expedient's communication with C-BAS, an API was developed. A detail for this API methods has been provided in Appendix I.

## 3.1.2 Workflows

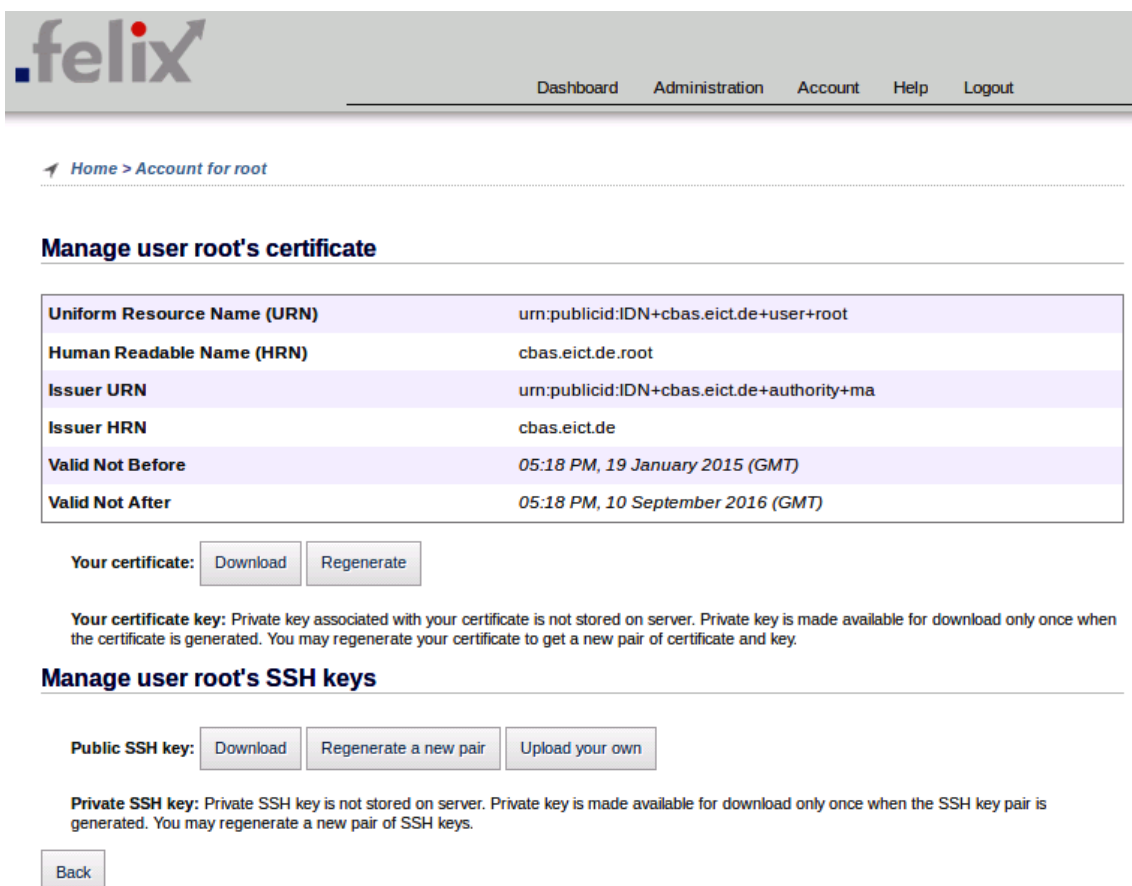
### 3.1.2.1 Interaction of Expedient with other modules

Figure 3.7 shows a sequence diagram of the interactions of Expedient with the user and other components like



The login page features the .felix logo at the top left. Navigation links for 'Log in', 'Register', and 'Password reset' are at the top right. A 'Log in' link is also present below the header. A 'Switch to username & password based login: Show' button is located above the certificate and private key upload sections. These sections each have a 'Browse...' button and a 'No file selected.' message. A 'Log in' button is positioned below these sections. At the bottom, there are links for 'Forgot your credential? Reset it!', 'Not a member? Register!', and 'Register!'.

Figure 3.5: Expedient's user login webpage



The management interface shows the .felix logo and navigation links: 'Dashboard', 'Administration', 'Account', 'Help', and 'Logout'. A breadcrumb trail reads 'Home > Account for root'. The main heading is 'Manage user root's certificate'. Below this is a table with certificate details:

Uniform Resource Name (URN)	urn:publicid:IDN+cbas.eict.de+user+root
Human Readable Name (HRN)	cbas.eict.de.root
Issuer URN	urn:publicid:IDN+cbas.eict.de+authority+ma
Issuer HRN	cbas.eict.de
Valid Not Before	05:18 PM, 19 January 2015 (GMT)
Valid Not After	05:18 PM, 10 September 2016 (GMT)

Below the table, there are 'Download' and 'Regenerate' buttons for the certificate. A note states: 'Your certificate key: Private key associated with your certificate is not stored on server. Private key is made available for download only once when the certificate is generated. You may regenerate your certificate to get a new pair of certificate and key.'

The next section is 'Manage user root's SSH keys'. It includes 'Download', 'Regenerate a new pair', and 'Upload your own' buttons for the public SSH key. A note states: 'Private SSH key: Private SSH key is not stored on server. Private key is made available for download only once when the SSH key pair is generated. You may regenerate a new pair of SSH keys.'

A 'Back' button is located at the bottom left.

Figure 3.6: Expedient's interface for user certificate and SSH keys management

the AAA and the RO to process different requests (this sequence diagram and the next ones appearing in this document are generated using [www.websequencediagrams.com](http://www.websequencediagrams.com)).

In order to access Expedient, the user has to login providing his certificate and private key. Expedient validates these data through AAA module and, if validated, user is logged in. Once logged, user can create projects and slices as required. The information about the projects and slices are stored in the AAA database together with the user privileges on them.

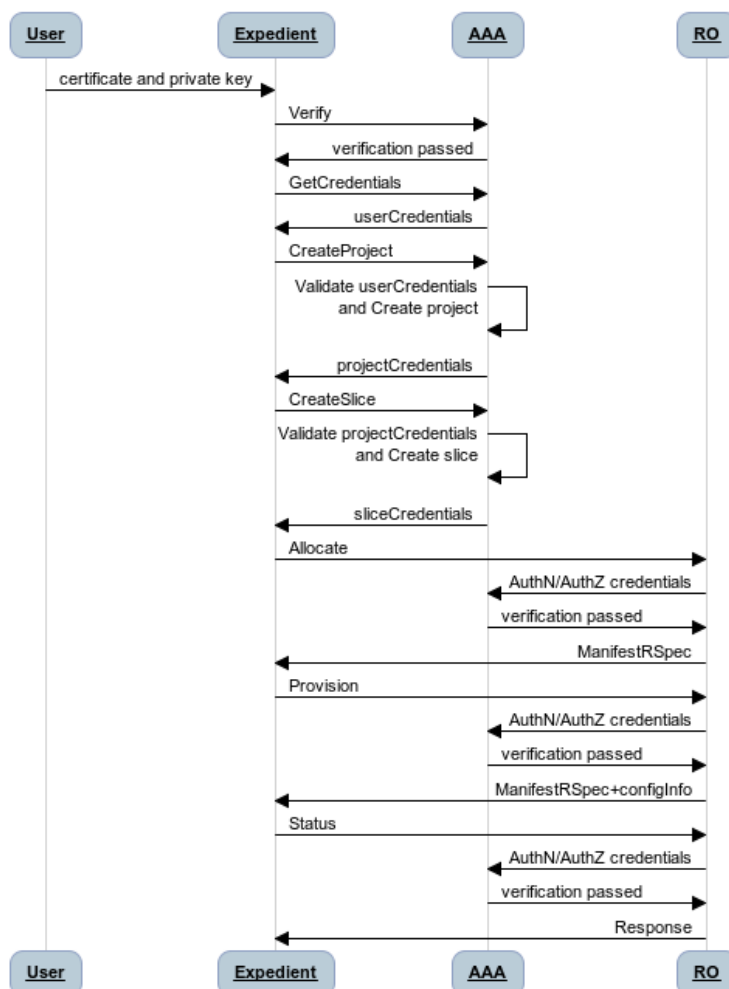


Figure 3.7: Expedient workflow

### 3.1.2.2 User Registration Process

Figure 3.8 below illustrates the user registration process in an island. In the first step the user registers himself with the registration portal by providing details, like first & last name, initial password, email address, affiliation, as well as a username. This registration request is forwarded to the system administrator for approval. If approved, the user is notified via an email containing the activation code. This step is necessary to verify the user email address. The User provides the received activation code at the registration portal to complete the registration process. Now the user is ready to log onto Expedient with a username and password. Every time a user logs onto Expedient, his credentials are fetched from C-BAS. First of such request is not fulfilled by C-BAS owing to the fact that the user was not yet registered with C-BAS. Therefore, Expedient has to register the user with C-BAS on his first logon. Assuming that registration succeeds, Expedient is returned with the user credentials, certificate and certificate key. As per policy Expedient does not store private keys locally therefore only the user certificate and credentials are cached for this user session. The user can request regeneration of his certificate and key for use



at Expedient login. This request is served by revocation of the existing certificate and issue of new certificate and key which is downloaded by user. From this point onwards, user logs onto Expedient via his certificate and key.

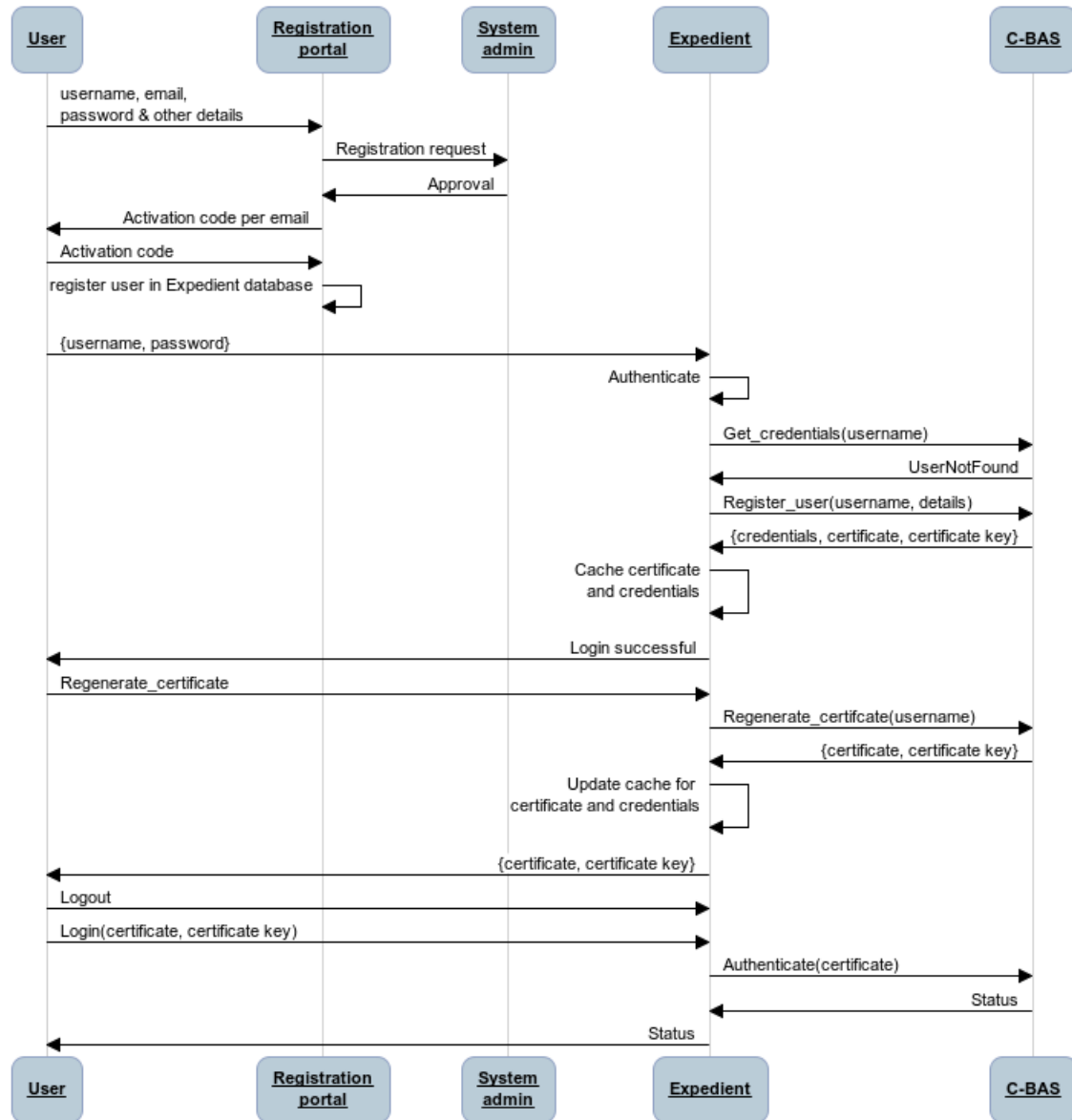


Figure 3.8: User registration process

### 3.1.3 Future Work

The Expedient GUI is a modified existing tool that required some modifications and extensions to provide the needed functionalities of FELIX project. Although it is a functional tool, it still requires further development to fulfill all the requirements. This is due to the fact that some developments were dependent on other components that were in process. One example of this is the RO plugin, which is dependent on the development of the RO.

Once the RO is finished, a GENI v3 API can be developed for the RO plugin so the RSpecs of the resources provided by the RO (TN-RM and SE-RM) can be sent and received. To interpret and generate these RSpecs, the corresponding parsers and crafters will be also developed.

Apart from this, in order to present the information of the TN-RM and SE-RM in the slice screen, this will have to be modified so these resources are visible in the topology panel and can be configured in the slice.

It is also expected to include the double topology (physical and slice resources) window presented in the monitoring section in the slice configuration window for the experimenters.

Finally, connectivity monitoring information described at the end of this section will be integrated in Expedient to provide users a vision of the status of the infrastructure and connectivity to help them select the proper resources for an experiment.

## 3.2 AAA

One of the main objectives in FELIX is designing a large scale experimental facility through federation that spans across continents and makes it possible to perform experiments at a global scale. The success of such deployment very much depends on the design and implementation of essential security and AAA mechanisms that not only ensure the robustness of the facility against intrusions but also ease experimentation and system administration in such complex environments. Despite of its importance, the research activities in this area, however, have only recently started and many challenges are still to be resolved. C-BAS is an initiative in this direction that proposes a secure and flexible certificate-based AAA architecture for SDN experimental facilities (SEFs). Advanced certificate-based authentication and authorization makes C-BAS inherently resilient against attacks specific to traditional AAA mechanisms, brings autonomy to system administration, and greatly smooths the process of federation.

To the best of our knowledge, C-BAS is the only open source AAA solution for SEFs that is actively being maintained & developed and freely accessible for whole research community. There are few other clearinghouse implementations that have been developed for particular SEFs and often restricted to consortium members. For example, Expedient[4] is GENI control framework with tightly coupled clearinghouse and, therefore, lacks the distributivity and flexibility. Similarly, ProtoGENI[5] clearinghouse has been based on a particular Slice Facility Architecture (SFA)[6] and designed to support only small sized federations. Likewise, GENI clearinghouse[7] with restricted dissemination policy implements its own clearinghouse API followed within GENI federation. In contrast, C-BAS access interface supports Common Federation API version 2 (FAPIv2)[8], set of standard APIs that any GENI-compatible Federation should offer. Moreover, C-BAS is flexible enough to support other APIs through plugins.

### Certificate-based authentication and authorization

Authentication (AuthN) is the process of verifying whether an entity is in fact who or what it claims to be. A certificate is a digitally-signed document which authenticates the identity of an entity such as, for example, a user, a piece of software, or a server. A certificate asserts the ownership of a public key to the named subject (owner) of the certificate. Certificates are typically issued by a third party called the Certification Authority (CA), which is trusted by both the owner of the certificate and the entity relying upon the certificate. This implies that if an entity trusts the issuing CA, it will also rely upon the identity assertion made by the private key that corresponds to the public key of this certificate. This lays the foundation of certificate-based authentication.

X.509 [9] is the most widely accepted standard for digital certificates. Each X.509 certificate has several information fields, including the distinguished name (DN) of the issuing CA, validity period, subject DN and its public key and, most importantly, the digital signature of the issuing CA. A digital signature is basically a hash of the data fields of the certificate encrypted with the signer's private key. The CA digital signatures are included in a certificate to ensure the integrity of its contents which involves the process of decrypting the digital signature using the CA public key and matching the data with the self-computed hash of the certificate contents. X.509 allows a CA to revoke issued certificates as a solution to problems such as the loss of the private key corresponding to the certificate. For this purpose, a revocation list is maintained and published by the CA.

An important step in the process of certificate-based AuthN is to validate the presented certificate. For a certificate to be considered valid, (i) the issuing CA must be trustworthy for the relying party, (ii) the certificate must be valid at verification time, and (iii) the certificate ought not to be revoked by the issuing CA. If all three

conditions are met, the entity provides a proof of being the legitimate owner of the presented certificate. In other words, the proof of possessing the private key corresponding to subject's public key in the certificate must be presented. This process is depicted in Figure 3.9. The user provides his private key to the user-agent (Figure 3.9:A), and the relying server provides some known data to the user-agent, which is encrypted using the private key (Figure 3.9:B) and is sent back to the relying party (Figure 3.9:C). If this encrypted piece of data (named evidence) can be successfully decrypted using the subject's public key, it verifies the entity's possession of the private key (Figure 3.9:D) and hence the claimed identity of the entity as asserted by the certificate is also verified (Figure 3.9:E).

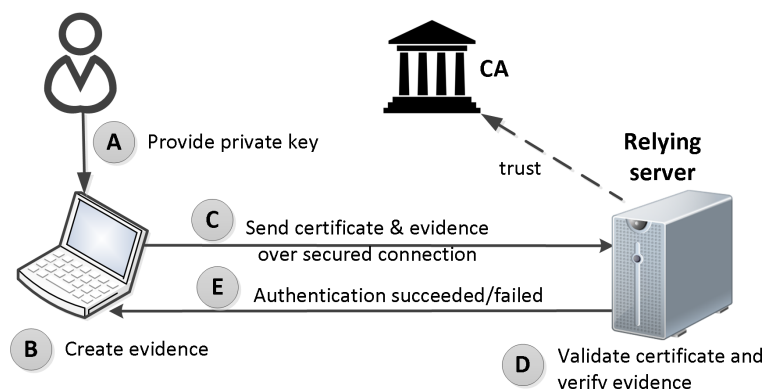


Figure 3.9: Certificate-based identity verification and authentication

AuthN alone is not sufficient to grant system access to a user and must be complemented by an authorization (AuthZ) process by which someone or something is allowed to perform certain operations or access particular information. AuthZ can be performed either by using a directory service like LDAP [10] where user privileges are listed against his identity or through digitally signed documents (credentials) that list user privileges on a target object. In principle, credentials are similar to certificates except that credentials assert privileges while certificates assert identities. A credential document contains the identity of the owner (the entity whose permissions are being specified) and the target (the entity on which the permissions are being specified), the validity period, list of privileges as well as the digital signatures of the issuing authority. If a certificate-based AuthN is used, it is advantageous to use an entity's certificate as part its identity in the credentials. This way, the identity of both owner and target can be reliably verified. In the context of FELIX, Slice Federation Architecture (SFA)[6] credentials format has been considered which is rather simple and supports a table-driven mapping mechanism between attributes and allowable actions.

**Member roles and privileges** Consider Figure 3.10, which illustrates member roles to determine the access level (or privileges) of a person or entity. This includes *Lead* who is the owner and principal contact regarding all activities on a project or slice. There is only one Lead per project/slice. (2) The *Admin* who is authorized to modify project/slice attributes as well as change the role of other members except the Lead. (3) A *Member* is a user that has read-write access on the entities managed by that group. (4) The *Auditor* is a member of the group with read-only privileges to monitor group activities. In addition, a *root* member role has been introduced to facilitate the execution of system administration tasks conforming to the scheme of privileges. This in turn also helps in logging and accountability of administrative actions. Another role has been defined for user-agents hosted by islands that need privileges to perform actions like, registration of new members and RMs etc. In contrast, the user-agents that run in user premises like OMNI[1] do not need credentials as they typically speak for their operating users. In some cases, an RM has to send a request for acquiring public SSH key of a member, but such requests are served without authorization.

**Delegated credential:** A user (delegator) can bestow all or parts of his privileges to another user (delegatee) by providing him with the delegate credentials. The delegated credentials are then used by the delegatee

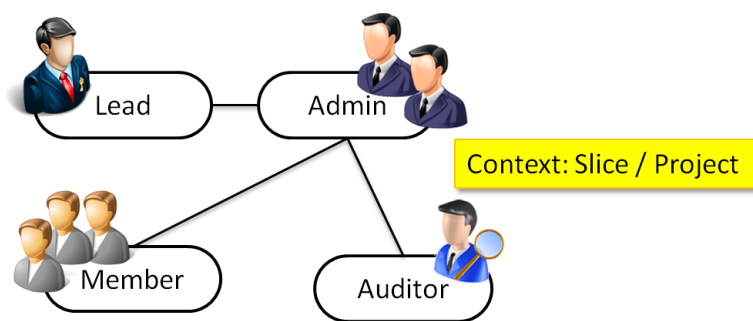


Figure 3.10: Member role hierarchy

along with his identity to get him authorized. In this case, only the delegatee is held accountable for his actions. The delegator is not accountable for delegatee actions. Delegated credentials must be digitally signed by the delegator.

**Speaks-for credential:** This is primarily intended for supporting tools that "speaks-for" their users. Speaks-for credentials allow a tool to get authorized using the user's privileges. Though the user is accountable for each action taken, the information about the tool used is also logged in the system. This way, speaks-for credentials allow for tracing which tool was used to perform any operation on the experimental facility.

**Speaks-as user:** If speaks-for credentials are not supported by a tool or user-agent, then the speaks-as concept can be employed. For this purpose, the user will have to provide his certificate and the private key to the user-agent. The private key is required by the user-agent to digitally sign the user's request so that they can appear as if they are directly coming from the user. In this case, the user is accountable for each action taken and is also logged as the action-taker. Moreover, the information about the user-agent is not logged.

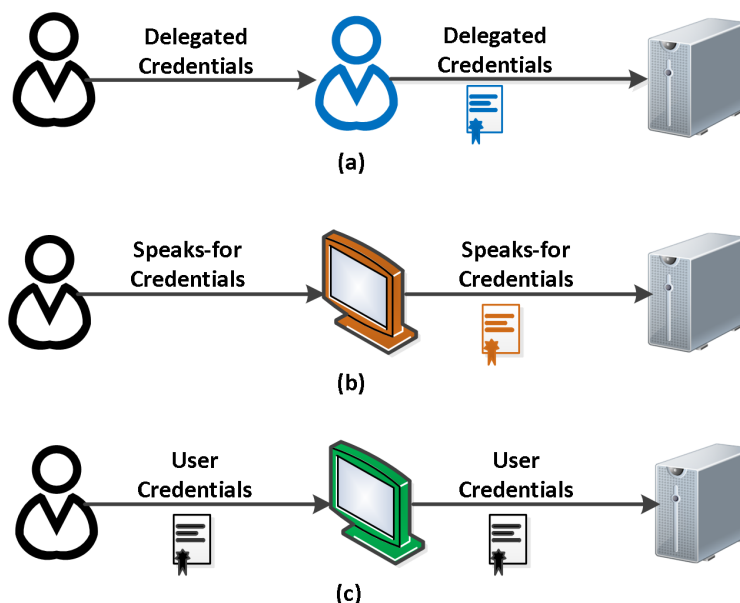


Figure 3.11: Delegation of privileges. (a) delegated credentials, (b) speaks-for credentials, (c) speaks-as user

**Clearinghouse services** The C-BAS ClearingHouse (CH) comprises a set of related services supporting AAA operations. CH is also a central location to lookup information about members, slices and other available services in the testbed. CH services can be categorized into three groups: (1) Registration and management services which

provide a lookup for available services in the facility and facilitate the registration of new members, projects and slice objects. (2) AuthN and AuthZ services that manage the credentials of all SEF entities and enforce predefined policies. (3) Accountability services which keep track of all transactions. CH services are offered with the help of the following functions and authorities.

The **Member Authority** (MA) is responsible for managing and asserting user attributes. It generates member certificates and credentials, which specify the attributes and roles associated with a member, while the certificate identifies the member within the credentials. The MA maintains a database of registered members and their associated information including, but not limited to, certificates and credentials, SSH (Secure Shell) and SSL (Secure Sockets Layer) keys as well as the human readable identity information like real name, institute, contact details and so on. MA is a central location to lookup and modify member information as well as register new members. In addition, MA should also maintain a Certificate Revocation List (CRL) which can be accessed by other entities for member certificate verification.

The **Slice Authority** (SA) creates and manages slice objects and the associated member credentials (called slice credentials). Slice credentials map member roles and privileges on a slice object, i.e., slice credentials authorize user actions at aggregates within a slice context. SA enables looking up slice credentials and supporting slice object operations such as "modify", "renew", "delete" and so on. The **Project Service** (PS) maintains a list of existing projects and asserts the member roles. The PS service can be SA-hosted and provides access for creating, looking up, updating, and deleting projects.

The **Service Registry** (SR) serves as the primary network contact point as it keeps a record of all available registered services such as SA and MA and offers their URIs.

The **Logging Service** (LS) provides for accountability by storing the transaction details between user-agents and aggregate managers. LS provides traceability between the slice and slivers and can be complemented with information such as the slice-associated project and Lead for full accountability.

The aforementioned CH services should be accessed via the Common Federation API (FAPI) [8]. The user-agents and RMs are to communicate with the CH through XML-RPC calls over a secured connection (SSL). By supporting FAPI, the CH makes itself compatible with command line interface user-agents like OMNI which are also capable of communicating with the aggregate managers using the GENI Aggregate Manager API [11].

### 3.2.1 Design

This section describes software architecture of C-BAS and its software components.

#### 3.2.1.1 eiSoil

The implementation of C-BAS is based on eiSoil ([www.eict.de/eisoil](http://www.eict.de/eisoil)), an open source light-weight framework originally developed for creating Aggregate Managers (AM). However its pluggable system and helper functions for common tasks make it perfect candidate to base C-BAS development on it. In short, eiSoil maintains a service registry or plugin-manager to host its plugin modules. This plugin-manager has been especially designed so that functionality implemented in one plugin module can be easily leveraged by other plugins. This architectural design is a main feature of eiSoil that offers service abstraction, modularity, extensibility as well as a clear separation of concerns. This way, use of eiSoil not only expedites the software development cycle but also makes software maintenance easier.

eiSoil, developed in Python, builds most of its functionality through plugins. From implementation viewpoint, an eiSoil plugin has a simple predefined code structure; source code files are organized under an individual folder along with two special files; (i) a JSON-style manifest document that describes the service provided by this plugin along with its dependencies on other modules and services, (ii) a Python coded initialization script that performs bootstrapping and registers the services of this plugin with the plugin-manager.

Out of the box eiSoil comes with a number of plugin modules covering a variety of helper functions. These plugin modules, termed vendor modules, are maintained by eiSoil development team. For example, *configdb* is

one of the vendor plugins which configures underlying SQL database that adds persistence to eiSoil. Similarly, *configrpc* provides means to perform configurations related to Remote Procedure Call (RPC). Another vendor plugin *geniv3rpc* implements RPC request handler for method calls of GENI v3 API[11]. Likewise, helper functions related to basic authentication & authorization are supplied through a plugin named *geniutils*. Moreover, plugins *mailer* and *scheduler* implement necessary support to send emails and perform scheduling to manage reservations.

Other prominent features of eiSoil include logging of error and debug information, scheduling and dispatching of asynchronous jobs, as well as, the ability to communicate with remote entities across the network.

### 3.2.1.2 C-BAS software components

As discussed C-BAS ([www.eict.de/c-bas](http://www.eict.de/c-bas)) is founded on eiSoil framework and, therefore, implements all of its functionality through plugins. Following eiSoil design philosophy C-BAS decouples its access interface from clearinghouse management API as depicted in Figure 3.12. This decoupling allows CBAS[12] to support multiple APIs (e.g., FAPI[8], SFA[6], FELIX[13] etc.) through code reuse without modifying overall architecture of the clearinghouse. The *delegate* module in Figure 3.12 acts as a translator between service access interface (e.g., FAPI) and the core clearinghouse methods. In addition, *delegate* also performs error handling by catching all clearinghouse errors and exceptions and re-throwing them after mapping onto access API specific counterparts. The *delegate* is also responsible for all kinds of namespace translations, for example, from resource URN to UUID etc. Most importantly, it is *delegate* which ensures that an API method call satisfies all requirements of authentication and authorization before forwarding it to the clearinghouse.

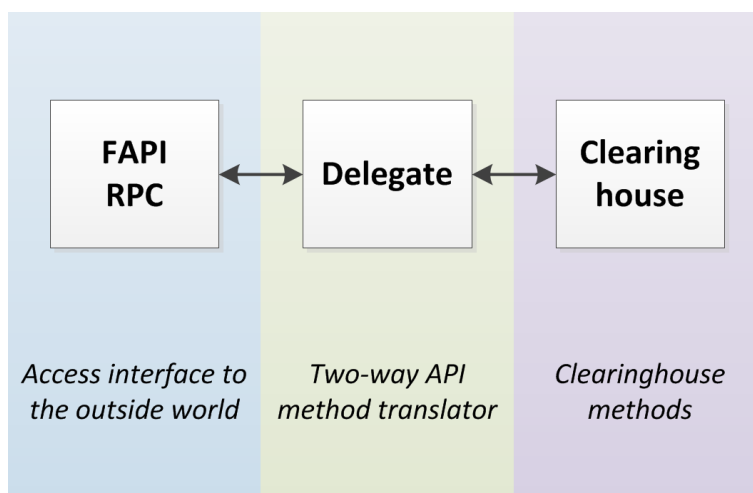


Figure 3.12: C-BAS service interfaces

In the following a list of C-BAS plugins is provided along with short description.

**fedrpc** has its name derived from 'Federation RPC API'. This plugin module builds the service access interface of C-BAS and is in charge of handling received XML RPC calls of FAPIv2. In addition to FAPIv2, C-BAS also supports other API methods, the details of which will follow later in this section.

**ofed** is an important plugin module that implements the *delegate* as shown in Figure 3.12. Owing to the fact that C-BAS supports multiple services like, Slice Authority (SA), Member Authority (MA), etc., for each service a dedicated delegate is implemented. This delegate not only maps FAPIv2 calls onto clearinghouse methods but also takes care of authentication and authorization. This is achieved through an extensive use of *geniutils* plugin, which is based on GENI provided library for SFA[6] styled certificate-based authentication and authorization. However, for use within C-BAS the aforementioned library has been extensively extended to support the rich functionality and roles & privileges system designed by C-BAS.



As implied from name **fedtools** is a plugin consists of helpers functions that are exploited by all plugins. These helper functions include conformation and validation checks for arguments passed in RPC method calls, credentials verification, and authorization checks for requested information. These helper functions are also utilized to get a mapping between member roles and their default set of privileges (see Table 3.1), as well as, to obtain the set of privileges needed to authorize a service access.

**omemberauthorityrm** implements Member Authority (MA) services of clearinghouse through management of members and SSH keys. It maintains a database of registered members, their credentials, certificates, and public SSH keys. Private keys corresponding to member certificates and private SSH keys are always handed over to their owners instead of making them part of the database. Moreover, it is **omemberauthorityrm** that registers new members and issues them certificates and credentials based on their assigned roles. It should be noted that MA maintains only system credentials which are not associated with any particular project or slice. These system credentials are mainly employed in tasks related to member information management and SSH key management by members themselves or by C-BAS system administrators. Being the anchor point for members information, **omemberauthorityrm** is consulted to update, lookup, and delete the member information and SSH key records. It is also often contacted by user-agents to retrieve member credentials and by AMs to request public SSH keys. The certificates and credentials issued to members bear a unique serial number and have a short validity time period after which membership must be renewed through **omemberauthorityrm**. A member certificate can also be invalidated following a certificate revocation process supported by **omemberauthorityrm**. For this purpose, **omemberauthorityrm** maintains Certificate Revocation List (CRL) which is periodically updated and disseminated. It should be noted that member registration, removal, and revocation are part of FELIX API that extends FAPIv2.

**osliceauthorityrm** plugin realizes Slice Authority (SA) services of clearinghouse. These services include management of projects, slices, and slivers. In addition, it also maintains user credentials for projects and slices. Project credentials are often used internally by SA to manage user membership for projects. On the other hand, slice credentials, which represent user membership for a slice and his privileges on that slice, are mainly required to authorize GENI API calls at AMs. **osliceauthorityrm** implements a number of functions to facilitate the creation, update, lookup, and removal of projects, slices and their memberships. For example, lookup of all projects or slices for a given member, members list of a project/slice, and credentials update when a member role is changed or its system membership is renewed. **omemberauthorityrm** also ensures that there is always exactly one Lead member for each slice/project who is the principal contact point for all activities of that slice/project. In contrast to MA, SA does not support membership revocation for slice/project. However, this can be achieved either through revocation of member certificate at MA, which would automatically invalidate any issued slice credentials or by removing user membership for the slice/project so that slice credentials would not be renewed for that user. By default slice credentials have a configurable lifetime of one month after which they must be renewed. **omemberauthorityrm** also supports delegation of slice credentials that allows a member to delegate some or all of his privileges to another member. Delegated credentials can be generated through SA provided API method. Once generated these credentials are returned to the user without storing them in the local database. Delegated credentials get invalidated if members certificates of involved parties expire or revoked and hence require regeneration.

**oregistryrm** implements Federation Registry (FR), sometimes also referred as Service Registry, which serves as primary contact point for C-BAS and its associated SEF. It keeps pointers to all C-BAS services (e.g., SA, MA) and AMs in the facility. Moreover, **oregistryrm** serves lookup requests for public certificate of authorities like SA and MA. All information available at **oregistryrm** is statically configured in a file named 'registry.json' during C-BAS setup. Owing to the fact that information distributed by FR is public, it serves all requests without any authentication and authorization. Pointer or URL of FR has to be shared with other federations or user-agents out-of-band as there is no other global service to gain access to FR.

**mongodb** realizes persistence for C-BAS by implementing a lightweight layer between clearinghouse and noSQL MongoDB[14] database. Any plugin which needs persistence leverages the helper functions offered by **mongodb**. It facilitates execution of database queries like create, update, lookup, and delete of entries in a col-

lection.

**registration** plugin handles new member registration requests sent by C-BAS registration client. It performs necessary sanity checks on such requests, processes them and returns member certificate and credentials back to user through C-BAS registration client. It should be noted *registration* is accessible only to C-BAS registration client while the user-agents with given privileges perform member registration through MA API calls.

Figure 3.13 is UML package diagram that lists constituent plugins of C-BAS and depicts their inter-dependency.

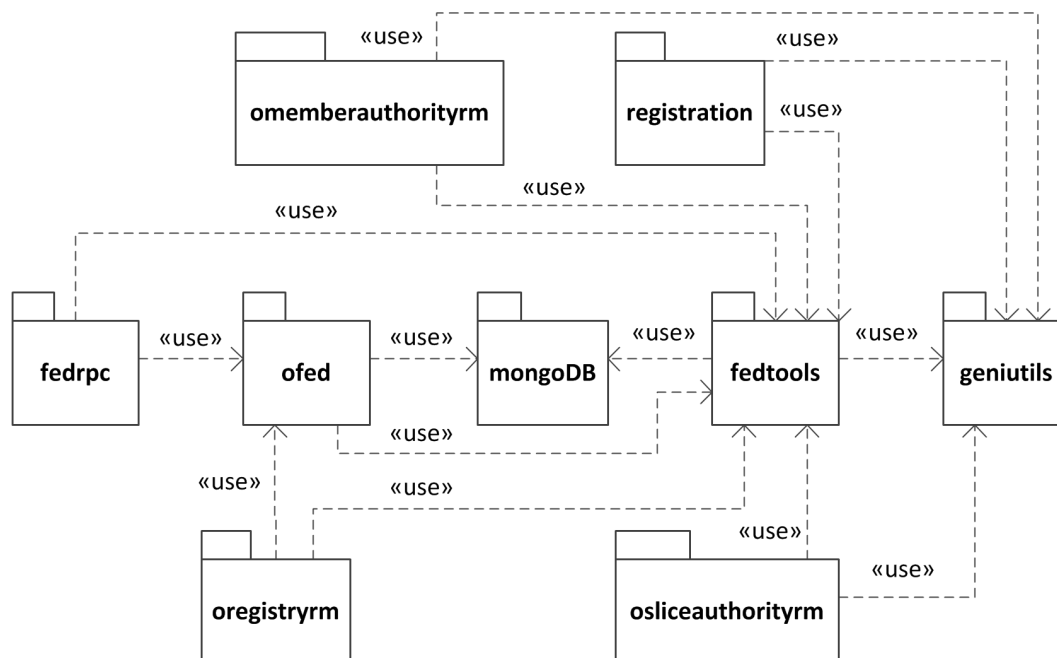


Figure 3.13: UML package diagram of C-BAS

### Roles and privileges

Table 3.1 shows the default privileges assigned to different member roles at their creation time. In addition to defaults, a member can be assigned with additional privileges through two ways; (i) through membership update where usually an Admin or Lead updates a member's privileges in C-BAS database, (ii) or through credential delegation process. Former is a long-term assignment which leads to new credential generation stored at C-BAS while the later is for short-term assignment where delegated credentials are generated with a limited time period validity. By policy, all default privileges can be delegated however a delegator may restrict further delegation of his delegated privileges. The concepts of delegated, speaks-for, and speaks-as credentials are shown in Figure 3.11.

### Membership revocation

C-BAS supports certificate revocation to withdraw certificates involved in incidents such as when private key of a certificate is lost or compromised, a certificate is mistakenly issued, or a member registration has to be confiscated. Certificate Revocation List (CRL) maintained and distributed by C-BAS enumerates revoked certificate (using serial number) along with the reason for revocation and the revocation time. In addition, CRL also bears its time of generation, next update time and the digital signatures of the issuing authority. CRLs distributed by C-BAS are of type X.509 version 2 encoded in PEM[15] format and conform IETF standards[16]. In future release, C-BAS is likely to get support for Online Certificate Status Protocol (OCSP)[17] that is considered a more reliable way of certificate status checking in real time.



Table 3.1: Member roles and associated privileges

Role	Context	Privileges
Auditor	Project	View, Monitor
	Global	View, Monitor
Member	Project	View
	Slice	View, Start, Stop
Admin	Project	View, Monitor, Update, SetAdminRole, AddMember, RemoveMember, ViewMember, UpdateMember, SetMonitorRole, CreateSlice, SlicesWildcard
	Slice	View, Update, SetAdminRole, AddMember, Start, RemoveMember, ViewMember, UpdateMember, Stop
Lead	Project	<Privileges of project admin>, SetLeadRole, Remove
	Slice	<Privileges of slice admin>, SetLeadRole
Root	Global	MembersWildcard, SlicesWildcard, ProjectsWildcard, RegisterMember, RenewMembership, RevokeMembership, RegisterService, ViewService, RemoveService,
User-agent	N/A	RegisterMember, RenewMembership, RevokeMembership, RegisterService, ViewService, RemoveService

### Certificate Chains and SSH Keys

MA and SA are expected to create and digitally sign the member and slice certificates and credentials. For this purpose the MA and the SA must be provided with the certificates that can be used to sign these digital documents. Such a certificate can be provided in the following way; the CH root certificate is used to create authority certificates for the MA and the SA. This way, the MA and the SA use their own certificates to create member/slice certificates and credentials. This provides each authority with its own identification in terms of certificate authority, and allows tracebacks for determining which particular MA / SA issued a certain certificate or credentials.

C-BAS needs to offer the user's public SSH key to the RMs in order to allow user login at virtual machines. These SSH keys are generated automatically during the member registration but a user may also provide his public SSH key during or after the registration process.

#### 3.2.1.3 API

C-BAS implements a complete set Common Federation API [8] methods. In addition, few more methods have been implemented for use in FELIX. Please refer to Appendix I for details.

### 3.2.2 Workflows

#### 3.2.2.1 Slice creation

In order to give readers an insight into C-BAS's request handling Figure 3.14 shows a sequence diagram of a use case where a slice is created through a user-agent. In order to create a slice in a project, the user has to first acquire his credentials for that project. For this purpose, a lookup is performed to fetch user's project membership information which includes his credentials. To authorize this lookup user's system credentials are presented which must pass through two checks; first check verifies if these credentials are valid and trusted while the second check ensures that user has sufficient privileges to access the requested information. Assuming that both checks pass, project membership info is looked up and returned provided user holds membership of the specified project. At this point user-agent can request for slice creation and present project credentials retrieved in previous step as means for authorization. These credentials are verified in the same way as described earlier and on success the requested slice is created and associated slice credentials are returned to user-agent for use in API calls to RMs.

The user who creates a slice is by default entitled with the Lead role of the slice. However a Lead member can willingly transfer this role to any other member of the slice. A similar procedure is followed for project creation.

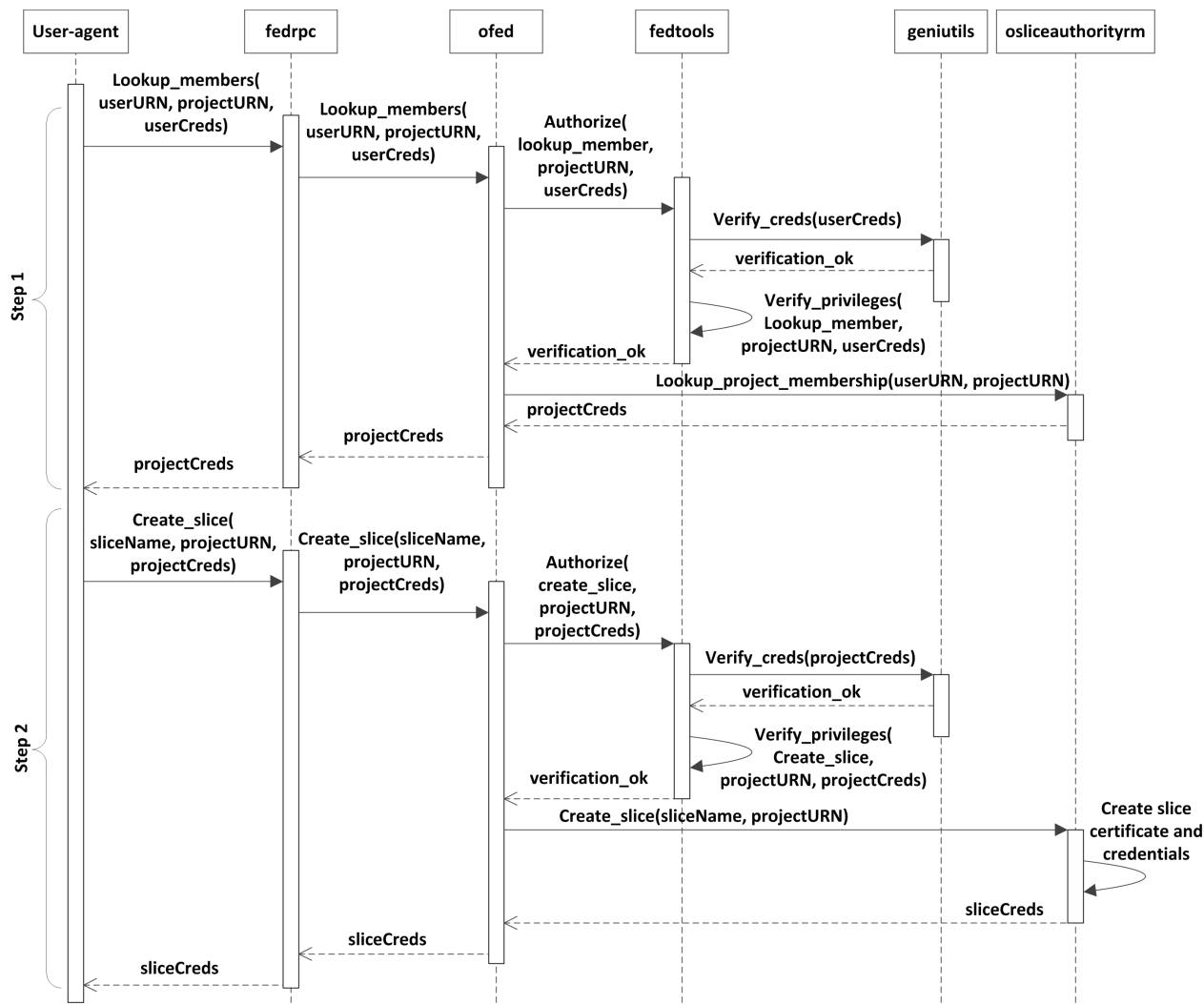


Figure 3.14: Project credentials lookup and slice creation processes

### 3.2.2.2 Resource allocation

Figure 3.15 demonstrates Expedient's interaction with C-BAS and an RM in a use case where an experimenter performs resource allocation. In the first step, the user requests a list of available resources using Expedient. For authorization purposes the user's system credentials are presented, which are verified by RM. Next, in order to reserve resources valid slice credentials must be presented at RM, however, the user must retrieve project credentials to authorize his request for slice credentials. Therefore, project membership lookup is performed to obtain project credentials and presented when requesting for slice credentials lookup. Once obtained, slice credentials serve the purpose of authorization for all GENI AM API calls to RM, like *allocate*, *provision* etc. It is encouraged that RMs should be equipped with necessary mechanism to authorize API calls themselves, however, C-BAS API also offers a method to do this task for an RM.

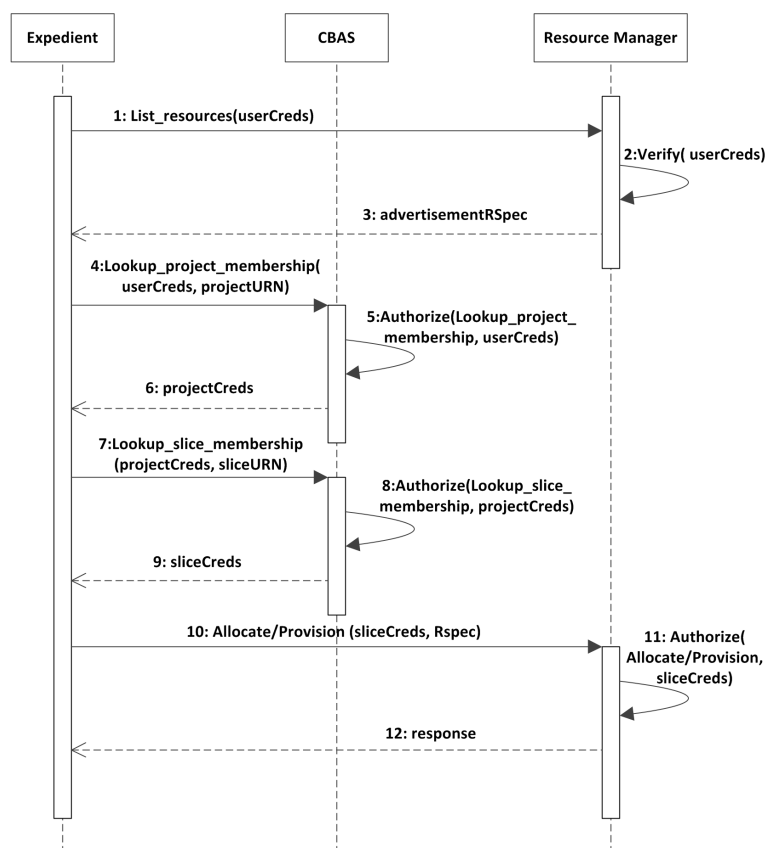


Figure 3.15: Expedient's interaction with C-BAS and RM to request resource allocation

### 3.2.3 Performance Evaluation

The performance of this C-BAS reference implementation has been evaluated through stress testing to judge its stability, scalability, and responsiveness. For this purpose, a set of operations was selected that encompasses all common operations expected to be performed on C-BAS in FELIX. The performance evaluation tests were executed on a virtual machine running Linux OS (Ubuntu precise 12.04.5 LTS) with a mere 2 GB of RAM. The server that hosts this virtual machine has a five years old Intel(R) Xeon(R) E5507 @2.27GHz CPU and employs KVM platform to host another six active virtual machines. The rationale behind not opting for a state-of-the-art hardware was to show that C-BAS imposes no stringent requirements for computational resources to support even a sizeable experimental facility. Although C-BAS supports multi-threading to process multiple requests in parallel and, therefore, has the capability to fully exploit today's multi-core CPUs, in this testing a configuration with single thread was chosen. In the following, a description and analysis is provided for four performance evaluation tests.

User registration operation is performed once for each new experimenter accepted to use the experimental facility. When received at C-BAS such requests are first put to authorization process involving certificate verification, credentials validation, and privileges check of the requesting entity. Figure 3.16 shows the results of stress tests where user registration requests are sent to C-BAS in batches of different sizes. The distributions of request processing times have been depicted in the form of box and whisker plots representing maximum and minimum values of the sample space. The test scripts and C-BAS are running on the same virtual machine therefore negligible network delays are involved as all requests get routed through local loopback interface. It can be seen authorization process takes a relatively fixed amount of time in the range of 50–75 ms and remains unaffected by the database growth. On contrary, the total registration processing time increases with the database growth of registered users. This is mainly because of operations related to database writing and indexing as performed

by noSQL MongoDB daemon. Other than database writing registration process also involves request authorization as described above, as well as, the creation of user certificate (1024 bit RSA key pair) & signed credentials. However processing time for these operations is obviously not influenced by database growth.

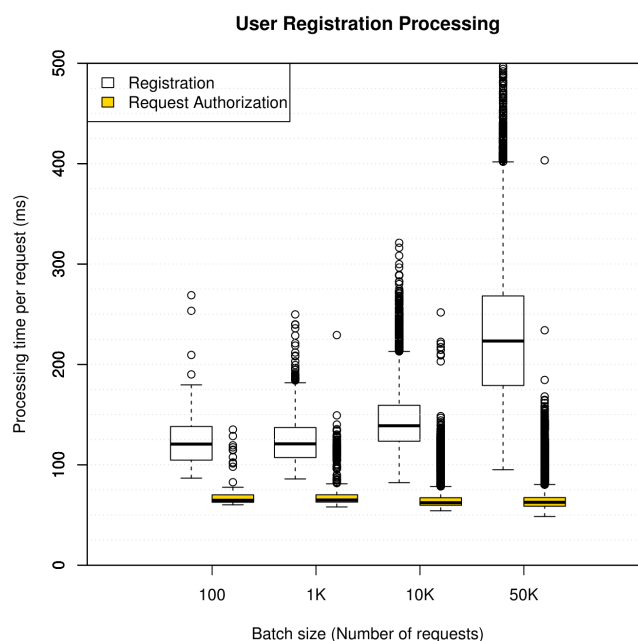


Figure 3.16: Boxplots of request processing times for user registration at C-BAS.

For FELIX experimental facility number of registered users are expected to fall in the range 1–10K and hence these users would experience registration delays with median from 120–140 ms. Though user registration is a onetime process performed for each new experimenter and is not a time critical operation, it is still handled efficiently by C-BAS.

Slice creation is an operation that is performed more frequently than the user registration. The project credentials needed to authorize slice creation request are looked up in Figure 3.14:Step1. The request to create a slice is then sent by user-agent and processed by C-BAS in Figure 3.14:Step2. The results of a stress testing that encompasses these operations are shown in Figure 3.17. C-BAS serves such requests quite well as evident from the near uniform distribution of response times with median value of 150 ms for slice creation and 200 ms for a combined operation of project credentials lookup and slice creation. Moreover, performance of C-BAS scales well as the number of created slices grows.

When a user logs onto Expedient, C-BAS has to provide its services for user authentication and credentials lookup as shown in Figure 3.4:5–10. Performance of these operations has been evaluated using a setup where another virtual machine sends 1,000 requests to C-BAS in a sequential manner. In 90 percent of these requests credentials of a random registered user are demanded while in rest of the cases the user whose credentials are asked for lookup does not in database. This makes such test more realistic by covering the cases where incorrect credentials are input at Expedient for user login. Moreover, evaluation test is performed for different sizes of C-BAS database to access their impact on the performance. In Figure 3.18 boxplots of request processing times for user authentication and credentials lookup have been presented. The authentication process (Figure 3.4:5–6) involves user certificate verification process including check against certificate revocation list. It can be noticed that user authentication process mostly executed within 50–60 ms while authentication and credential lookup collectively (Figure 3.4:5–10) takes 65–80 ms for completion. In addition, the request processing time is marginally affected by the database growth.

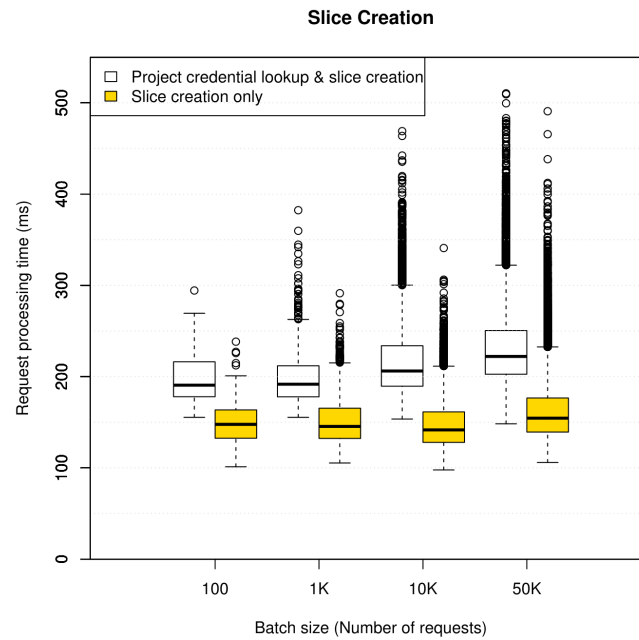


Figure 3.17: Boxplots of request processing times for project credentials lookup and slice creation.

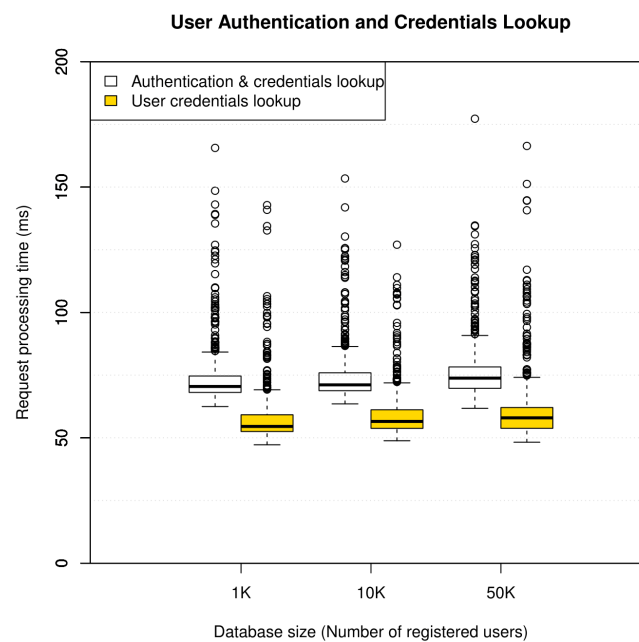


Figure 3.18: Boxplots of request processing times for user authentication through certificate verification and user credentials lookup.

Project and slice credentials lookups are among the most frequent requests processed by C-BAS. For example, slice credentials are included in almost all GAPI call to AM as means for authorization (see Figure 3.15). Performance evaluation of project and slice lookup is carried out in a similar manner as explained for previous evaluation test and the resulting measurement distributions have been demonstrated in Figure 3.19 as boxplots. It is evident that C-BAS can lookup credentials of a random slice (Figure 3.15:4–6) in a database of up to 50K slices

in less than 75 ms. Similarly, project and slice credentials lookups (Figure 3.15:4–9) are collectively processed with a median value well below 150 ms. Furthermore, it is apparent that project and slice credential lookup processing time is marginally affected by database size. With such performance C-BAS can carry out approximately 15 lookups per second in a sizeable database of 50K registered slices without complaining about limited computational resources.

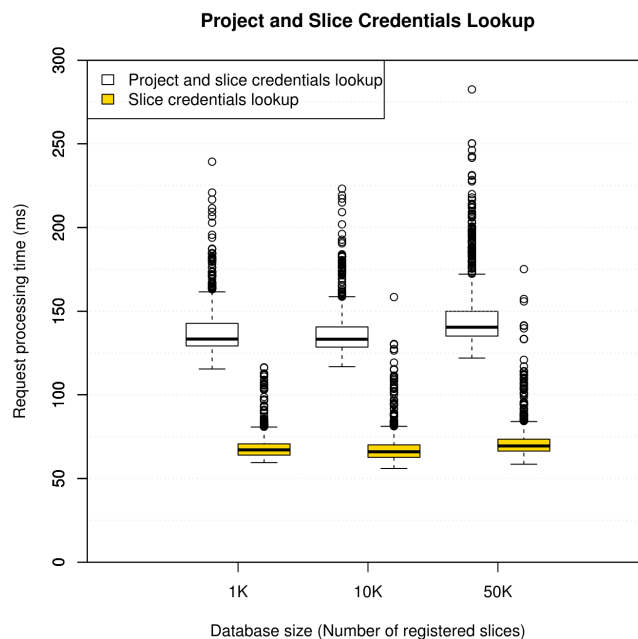


Figure 3.19: Boxplots of request processing times for project and slice credentials lookup at C-BAS.

### 3.2.4 Future Work

After the core functionality implementation of C-BAS and extended FAPIv2, next steps should focus on its integration with other components in FELIX architecture, e.g., RO, RM, stitching entity. In addition, C-BAS should also fulfil the needs of monitoring infrastructure for authentication and authorization mechanisms. Furthermore, C-BAS has to incorporate token-based authentication and authorization mechanism for use with NSI framework in transit networks.

## 3.3 Public Monitoring

Slice monitoring as described in D3.2 [18] is used to collect and aggregate monitoring data from resources and slices. That data, such as bandwidth measurement and cpu load, is used both for slice provisioning and offering status monitoring to the user once a slice has been setup. The monitoring data is time stamped and stored in one or more databases and aggregate as well as exchanged between several master monitoring systems (e.g., one in Japan and one in Europe).

Apart from slice monitoring, we want to provide on the public web site of FELIX an overview of available infrastructure, that is, connectivity and computing resources, and this with live status updates. The goal of the public monitoring application is to collect data for this overview. This data is mostly status information as retrieved from the Resource Manager components, and possibly resource-specific measurements for infrastructure that is not managed by an RM (for example, ping results on support servers such as DNS, VPN). Unlike slice monitoring, this data is not directly monitored from the resources (such as switches, virtualisation servers), which is why the

public monitoring application is a stand-alone tool not dependent on the slice monitoring infrastructure. Since the volume of monitoring data and status information is minimal, public monitoring is performed in a centralised manner and does not need aggregation.

### 3.3.1 Design

The main status information consists of switch status and link connectivity, and virtualisation server status. This information is retrieved periodically from SDN-RM and C-RM respectively, and this for each island. The information is stored in a light-weight database and retrieved and rendered into a web page when a visitor requests one of the island web pages from the public monitoring application. The application integrates with resource managers as shown in Figure 3.20.

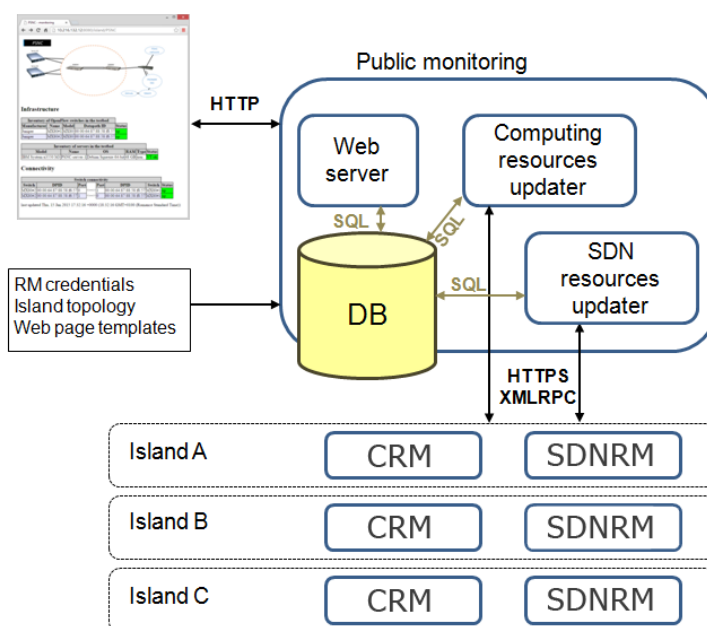


Figure 3.20: Public monitoring application interactions

The public monitoring application itself is a multithreaded Python application that runs a webserver for rendering web pages, a database to store resource status information, and one or more updater thread that each manage a specific type of resources. In the current implementation, there is an updater for computing resources (talks to C-RM) and for SDN resources (talks to SDN-RM).

#### 3.3.1.1 Web server

The web server in the application uses the web.py library. It constructs status information for a specific island using template files and static images for the island's topology. For each type of resources, there is a small amount of code that transforms the status information (from the database) into a human readable web page snippet (html), which is then included in the island's web page. The web server uses HTTP. Templates and styles can be configured for easy integration into another web site (i.e., the public FELIX website).

#### 3.3.1.2 Database

A light-weight SQLite database is used, as provided by the web.py library. The purpose of the database is to time stamp and store the latest status information for each type of resource, and for each island. SQL queries allow to quickly retrieve all information relevant to a specific island. No historical status information is stored. Because

of that the database is quite small so using SQLite is not a problem. The database is backed by a local file, and is not persistent (database and schemas are recreated each time the monitoring application is run).

### 3.3.1.3 Settings

Settings that must be persisted are stored in a local file (not in the database). These settings include program startup options, a list of islands, topology (switches, links) and server information, and the credentials used to communicate with the Resource Managers. Topology and server information is included because it allows to list resources to a visitor even if those resources have never been reported by a resource manager (typically because the resources have not been available since the public monitoring application was started) -- of course those resources would be reported as 'down' or unavailable to visitor. The data may also be used to suppress some of the resources reported from RMs, so they are not announced publicly.

### 3.3.1.4 Updater

Each type of resource has its own updater, in a separate thread. The updater implements the xmlrpc calls specific to that resource (and RM). Currently an updater for SDN resources and computing resources is available.

## 3.3.2 Workflows

On Figure 3.21, we show the workflow for requesting an island status page from the web server component. The web server receives a HTTP GET request, with an url `/island/<island>` corresponding to a certain island. `<island>` is used in collecting status information from the database. The web server thread requests information regarding switches (up/down), links (detected/not detected) and servers (up/down). The `describe_switches`, `describe_links` and `describe_servers` code snippets format the information returned by the database into readable text (html formatted). As a final step, all html fragments are rendered into an island page template and finally sent as html content to the visitor.

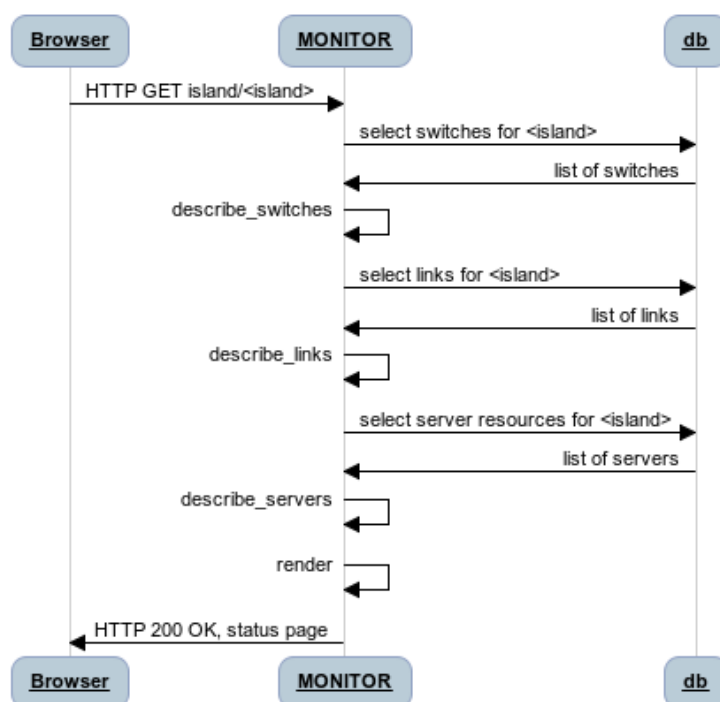


Figure 3.21: Requesting island status page



Figure 3.22 and 23 show the work flows for the SDN and computing resources. By default, these updates occur every 2 min. There is a single thread (OFThread) which handles all SDN related xmlrpc calls (for all islands), and vice versa for the VTThread which handles computing resources.

At application start, a list of islands, switches and links is loaded from a settings file and inserted into an (empty) database. These settings include the credentials and urls for each island's SDN-RM. The SDN resource updater is then started as a new thread (OFThread), which updates all islands (SDN-RMs) sequentially. The updater uses the xmlrpc `get_switches` and `get_links()` API to retrieve current SDN status. The `get_links()` connectivity information in SDN-RM is constructed from information available through FlowVisor, and the actual connectivity itself is discovered through LLDP (LLDP packets sent by FlowVisor on its connected switches). After that, the updater loads the last updated status of the island's SDN resources from the database, and compares this with information retrieved from SDN-RM. The database information will be updated if needed. If SDN-RM reports new resources that were not in the database, they will be added (inserted) into the database. If the database contains resources that for some reason do not (or no longer) appear in the SDN-RM report, that resource will be set as unavailable in the database.

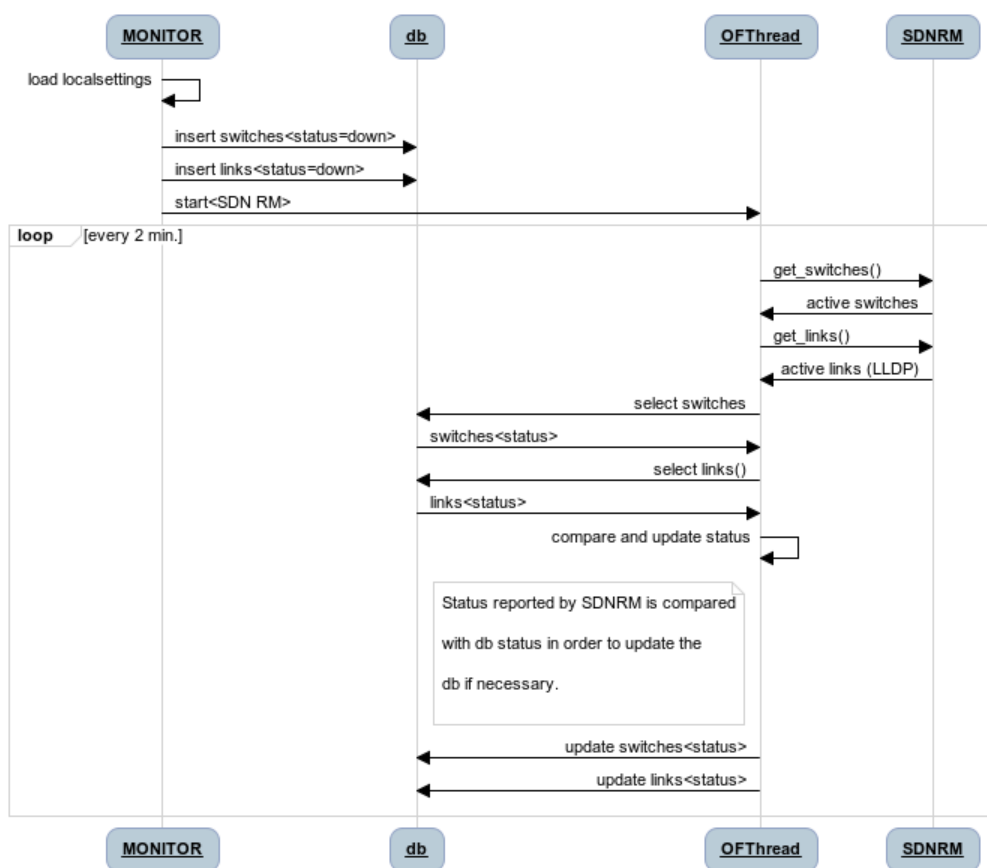


Figure 3.22: Updating SDN resources

The computing resources updater is similar (but simpler) to the SDN resource updater. Instead it uses the C-RM xmlrpc API, and only reports one type of resource (server), not multiple (switch/link). The updater is again initialised from local settings data, and runs in a separate thread (VTThread).

### 3.3.3 Future Work

The public monitoring application may need to be adapted to any API changes or extensions of resource managers. Also, if we wish to expose information publicly from additional (other types) of RMs, an updater should

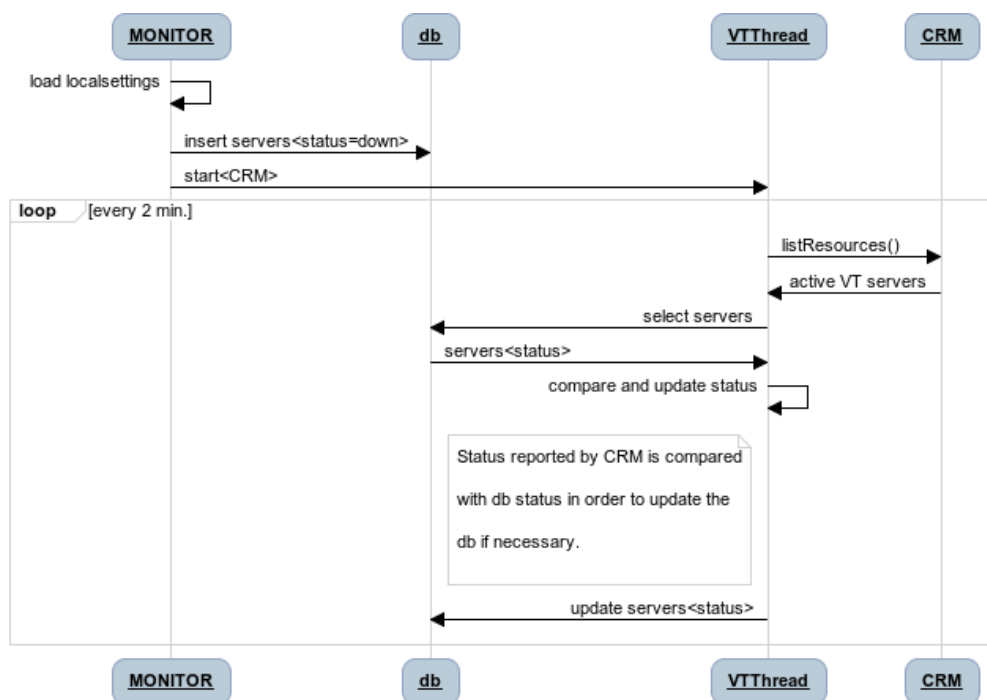


Figure 3.23: Updating computing resources

be added that implements the API for those RMs and transforms its status reports into database storable information.

Since the web page is publicly available, we may collect visitor feedback in order to improve the representation of status information on the monitoring web pages; also, the templates may receive small updates to match the styles of any embedding web site (such as the FELIX web site).

Finally, we may add resource types that are not managed by an RM, for example separate support servers such as file servers, VPN, DNS servers etc. which require other types of action to collect status information (e.g., ping, DNS query test...). This requires adding a resource updater as well as code that can describe the resource as a html snippet.

## 4 Deployment

This section provides the basic instructions to install and configure the previously described modules. For each module, the required dependencies are listed and then the steps to obtain the code, install it and configure it. Finally, a summarized operation manual is provided so the users can manage the modules.

### 4.1 Expedient

Expedient is a module of the OFELIA project control framework (OCF) [19]. OCF provides versioning tool called OFVER which allows installing, updating and configuring OCF components. The process is similar as explained in deliverable D3.1 [20] for C-RM and SDN-RM.

#### 4.1.1 Requirements and Dependencies

This is the list of requirements and dependencies to install Expedient. It is the basically the same as the needed for other OFELIA Control Framework components (e.g. Virtualisation AM, OpenFlow AM) and thus almost the same for the FELIX RMs.

##### 4.1.1.1 Requirements

The working environment must be a Debian-based distribution. Specifically, the framework is ensured to work under Debian 6 (Squeeze), but has been gradually being ported to Debian 7 (Wheezy).

Besides, Expedient interacts with different RMs and need some to be installed along it. This is the case of the C-RM module.

##### 4.1.1.2 Dependencies

There is a number of packages required for Expedient to work. These are retrieved either by Debian's Advanced Packaging Tool (*apt-get*) and the Python version system (*easy\_install*, *pip*).

#### Advanced packaging tool

The following Debian packages must be installed on the system for Expedient to properly work:

```
openssl, ssl-cert, libapache2-mod-wsgi, libapache2-mod-macro, python-setuptools,
apache2, python-django, python-mysqldb, python-ldap, python-openssl, python-m2crypto,
python-dateutil, python-decorator, python-paramiko, build-essential, python-imaging,
python-django-registration, python-configobj, python-pyparsing, python-lxml
```

#### Pip

The following python packages must be installed on the system for Expedient to properly work:

```
django-evolution (<=0.6.9), django-autoslug, django-extensions (<=1.2.5),
django_auth_ldap (==1.0.19)
```

#### 4.1.2 Configuration and Installation

##### 4.1.2.1 Installation

Save the OCF branch under folder /opt/felix:

```
mkdir -p /opt/felix/ocf
git clone https://github.com/dana-i2cat/felix.git /opt/felix/ocf
git checkout ocf
```

Now, the root chooses which modules to install through a screen with a menu:

```
cd /opt/felix/ocf/deploy
python install.py
```

The module to be selected in the menu is named *expedient*.

Once the installation starts, the OFVER installation scripts under the GUI (or *expedient*) folder are triggered and will ask whether the current installation is run within the OFELIA project or not. Select No (N) for non-OFELIA testbeds.

There are similar procedures for upgrading and removing the modules as well.

#### 4.1.2.2 Configuration

##### General parameters

FLAG	Values	Comments
ISLAND_NAME	String	Island/testbed name. Shown on the upper right corner of the Web frontend
SITE_DOMAIN	String	Expedient host domain name

Table 4.1: Configurable parameters

##### Root (Island Manager) account information

FLAG	Values	Comments
ROOT_USERNAME	String	Expedient's root username
ROOT_PASSWORD	String	Expedient's root password
ROOT_EMAIL	String	Expedient's root email. This is used to send notifications

Table 4.2: Island manager account parameters

##### Database parameters

FLAG	Values	Comments
DATABASE_USER	String	MySQL username
DATABASE_PASSWORD	String	MySQL password
DATABASE_HOST	String	MySQL host (e.g. 127.0.0.1)
DATABASE_NAME	String	Expedient database name

Table 4.3: Database configuration parameters

### 4.1.3 Operation

As commented previously in this document, Expedient provides functionalities to both testbed administrators and experimenters users. This section provides a short step by step guide the main functionalities.

#### 4.1.3.1 Experimenter operations

##### Create project

- In the Dashboard page, under Projects list, press "Create" button.
- Provide a name, organization, an approximate end date and a description for the project and press "Request" button.
- Wait for project approval.
- Once the project is approved, it will appear in the Project table of the Dashboard screen.
- Inside a project page, the user can Add Members, Add Aggregates and Create Slices.

##### Add Members

- In the Project page, press "Add Members" button.
- From the dropdown list, select the user to add to the project.
- Select the roles that the new member should have in the project. Owner users can add members to the project, researchers cannot.
- Press "Add" button.

##### Add Aggregates

To be able to create slices that can reserve resources on aggregates, the user will need to add aggregates to the project.

- In the Project page, press "Add Aggregates" button.
- From the list of aggregates available in the testbed, select the ones to add to the project.
- When the selection is finished, press "Done" button. The added aggregates will appear in the Aggregates list of the project page.

##### Create slice

- In the Project page, press "Create Slice" button.
- Provide a name and a description for the slice.

- Press "Save" button.
- In the slice page, press "Add an Aggregate Manager to the current slice" button.
- From the available AMs in the project, select the ones to use in the slice.
- Press "Done" button. The resources of the added AMs will appear in the Physical topology window of the slice page and the inferior panels will include the options to configure them.
- Once configured the resources as explained in deliverable document D3.1 [20], press "Start Slice" button on top of the page.

#### 4.1.3.2 Administrator operations

##### Add Aggregate

Administrators can add new resources by adding their Aggregate Manager.

- At the bottom of the Aggregate Managers panel, select the type of aggregate to add from the dropdown list.
- Press "Add Aggregate" button.
- In the new window, provide the required data to add the AM (name, user, password, url, etc.) and press "Create" button. The new AM will appear in the AM list.

##### Manage permissions

- In the Manage Permissions panel there is a list of requested permissions from the users (create project, etc.).
- Administrator can select to approve or deny each request.
- Press "Save" button to apply the decisions.

##### Manage users

Administrator can create, delete, view or edit users.

- Click on "Manage Users" in Users & Permissions panel.
- New page shows a list of existing users. On right column, administrator can choose view/edit the user info or delete it.
- At the bottom of the page, the administrator can provide the info to create a new user.
- Once the form is completed, press "Create user" button.

##### View logs

In the monitoring panel, there is a list of the available logs of the different AMs or components.

Administrator can view their content or clear it with the options on the right column of each available log.

##### Plug-ins

At the Plug-ins panel the administrator can perform specific operations for different plug-ins added to the CF. This function depends on the plug-in. E.g. Remove VMs from expedient side by providing the VM ID.

New plug-ins can provide new functionalities.

## 4.2 AAA

All functionality related to authentication & authorization in FELIX islands is handled by C-BAS. In the following guidelines about C-BAS deployment are provided.

### 4.2.1 Requirements and Dependencies

C-BAS has both Linux OS and Python packages dependencies.

#### 4.2.1.1 Linux Packages

- *Swig* (needed for the M2Crypto Python package)
- *MongoDB* database running on the local host and default port
- *Python-pip* and *Python-dev* are required to install Python packages in next step
- *Xmlsec1* used to digitally sign XML documents

#### 4.2.1.2 Python Packages

C-BAS requires following Python packages

Flask, Flask-XML-RPC, Jinja2, M2Crypto, MarkupSafe, SQLAlchemy, Werkzeug, argparse, blinker, cffi, cryptography, flup, itsdangerous, jsonrpclib, pyOpenSSL, pyRFC3339, pycparser, pymongo, python-dateutil, pytz, six, wsgiref.

These dependencies can then be installed using following command

```
pip install -r requirements.txt
```

### 4.2.2 Configuration and Installation

Copy the following configuration files and adjust the entries as required

```
cp deploy/config.json.example deploy/config.json
cp deploy/registry.json.example deploy/registry.json
cp deploy/supplementary_fields.json.example deploy/supplementary_fields.json
```

Generate certificates and credentials by running shell script available at

```
test/creds/gen-certs.sh
```

Copy Expedient's credentials to FAPI credentials directory, i.e.,

```
cp test/creds/expedient*
/opt/felix/expedient/src/python/expedient/clearinghouse/fapi/creds/
```

### 4.2.3 Operation

Fire up the C-BAS server from Ohouse directory

```
python src/main.py
```

Enable C-BAS in Expedient's settings file under

```
/opt/felix/expedient/src/python/expedient/clearinghouse/defaultsettings/cbas.py
```

Restart Apache server Now login to Expedient using administrator's username & password and download your certificate & private key available under Account->Manage certificates. With certificate and private key downloaded, logout of Expedient and login again choosing certificate-based login option. Log file of C-BAS is generated under log/eisoil.log Log file that covers interaction between C-BAS and Expedient, if enabled, should be available at

```
/var/log/apache2/cbas.log
```

## 4.3 Public Monitoring

The public monitoring web page gathers connectivity and resource status information from several sources and provides a per-island overview. Only a single instance needs to be deployed, although alternative deployments with a different url and/or configuration are of course possible

### 4.3.1 Requirements and Dependencies

The monitor application is based on Python, using the web.py library for serving the public web page and database backend. urlparse, xmlrpclib and etree are used to communicate with the Resource Managers, json and urllib2 are used to communicate with Emulab monitoring tools. To install web.py (Debian):

```
apt-get install python-webpy
```

### 4.3.2 Configuration and Installation

Save the monitoring branch under folder /opt/felix:

```
mkdir -p /opt/felix/
git clone https://github.com/dana-i2cat/felix.git /opt/felix/
git checkout monitoring
cd /opt/felix/modules/monitoring/public/src
```

Configuration of the application contains a list of RM location/credentials, partitioned by island. Normally, the monitor application will show status for all resources that have been reported at least once by an RM, but single resources such as SDN links can also be listed explicitly in the configuration file. This allows the monitor application to know about these links even if they have never been reported (since application start), and to display such resources as down or unavailable. The configuration details are found in *localsettings.py*. The repository contains a *localsettings.py.EXAMPLE* file.

New resources or RMs can be added to this file as they become available. The file provided contains the configuration for available FELIX resources.

The island pages are based on web.py templates found in the *templates/* subdirectory. To add a specific layout or style to the public monitoring pages, the *templates/island.html* can be changed.

Static content such as the island description must be added to the *static/* subdirectory, where *<island>.png* is the island topology shown on the respective island page. Additional static files (such as .css) can be added here as well, if for example they are needed after changes to the templates.

### 4.3.3 Operation

Running the monitor application:

```
python ./main.py
```

During runtime, a sqlite monitor.db file will be created. This file is non-persistent and will be recreated at startup (this includes creation of db schemas). Permanent configuration data is stored in the localsettings.py file. Resource status is not persisted, as it is fetched continuously from RMs.

From the user side, summary web pages are available for each island, as shown on Figure 4.1, on the url



<http://157.193.215.150:8080/island/<islandname>>

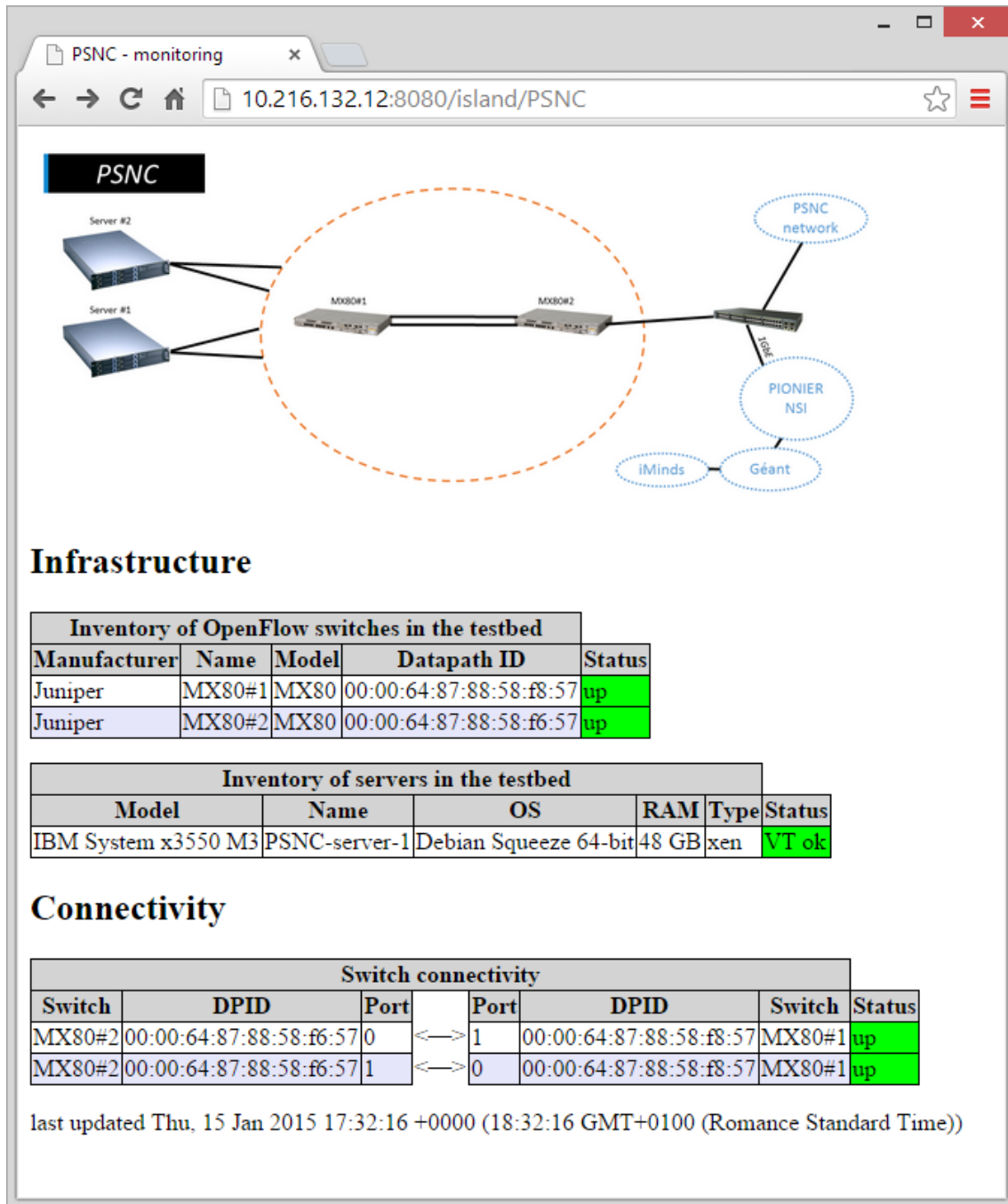


Figure 4.1: Example public monitoring web page (PSNC)

## 5 Conclusions and Summary

End user tools are the visible part of any IT project. They interact with the users and provide access to all the functions in an easy and friendly way. If an user tool does not accomplish this, regardless of how good and efficient is the implementation of the project, the infrastructure will fail to be utilized to its full potential. This document has presented the adaptation of OFELIA Expedient, an existing GUI-based user-agent which delivers most of the functionalities required by FELIX with a number of modifications.

Furthermore, a view on the architecture of authentication and authorization mechanism and its integration with Expedient has been provided under the description of C-BAS. C-BAS is a certificate-based authentication mechanism proposed as an alternative to traditional password-based authentication methods that usually have limitations in design and vulnerabilities to common attacks. This document reports the implementation level details of C-BAS as well as evaluate its performance through stress testing. The obtained results provided an evidence of C-BAS's computational efficiency, scalability, and fitness for FELIX's large scale experimental facility.

Finally, the document presented an overview of public monitoring tool which has been integrated with project website. It offers a general live view of the infrastructure status available in FELIX experimental facility to help potential users know which resources are at hand for their intended experiment.

## References

- [1] "The Omni client." <http://trac.gpolab.bbn.com/gcf/wiki/Omni>.
- [2] R. Krzywania, et al., "General Architecture and Functional Blocks," FELIX Deliverable D2.2, Dec. 2013.
- [3] M. Sune, et al., "Design and implementation of the OFELIA FP7 facility: The European OpenFlow testbed," *Computer Networks*, 2014.
- [4] J. Naous, et al., "Expedient: A centralized pluggable clearinghouse to manage geni experiments," Jan. 2010.
- [5] "Protogeni website." <http://www.protogeni.net>.
- [6] "Slice-Based Federation Architecture 2.0." <http://groups.geni.net/geni/attachment/wiki/SliceFedArch/SFA2.0.pdf>, July 2010.
- [7] "GENI Clearinghouse." <http://groups.geni.net/geni/wiki/GeniClearinghouse>.
- [8] "Common Federation API." <http://groups.geni.net/geni/wiki/CommonFederationAPIv2>, Nov. 2013.
- [9] ITU-T Recommendation X.509, "The Directory: Public-key and attribute certificate frameworks," version 3, Oct. 2012.
- [10] "Lightweight Directory Access Protocol (LDAP): The Protocol," IETF RFC 4511, Jun. 2006.
- [11] "The GENI Aggregate Manager API." <http://groups.geni.net/geni/wiki/GeniApi>, 2013.
- [12] U. Toseef, et al., "C-BAS: Certificate-based AAA for SDN Experimental Facilities," in *Proc. EWSDN*, Sept. 2014.
- [13] R. Krzywania, et al., "Experiment Use Cases and Requirements," FELIX Deliverable D2.1, Sept. 2013.
- [14] "MongoDB database." <http://www.mongodb.org/>.
- [15] "Privacy Enhancement for Internet Electronic Mail," IETF RFC 1421--1424, 1993.
- [16] "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile," IETF RFC 5280, May 2008.
- [17] "X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP," IETF RFC 6960, June 2013.
- [18] T. Ikeda, et al., "Slice Monitoring," FELIX Deliverable D3.2, Jan. 2015.
- [19] "OFELIA Control Framework (OCF)." <http://fp7-ofelia.github.io/ocf/>.
- [20] C. Fernandez, et al., "Resource Planning and Provisioning," FELIX Deliverable D3.1, Jan. 2015.

## Appendix I

### A C-BAS API methods

C-BAS is fully compliant with GENI Common Federation API version v2 [8]. In addition, C-BAS also supports following method calls.

#### A.1 Slice Authority (SA) API methods

```
"""
```

Verifies if given slice credential is valid. If successful, a list of privileges embedded in given credential is returned. This method also supports verification of delegated credentials.

Arguments:

creds\_to\_verify: Slice credentials to be verified in SFA string format.  
 cert\_to\_verify: Slice certificate  
 target\_urn: Slice URN  
 certificate: Certificate of the method calling entity  
 credentials: Credentials of the method calling entity

Return:

If successful, a list of privileges provided by slice credentials in string format. Otherwise, error code.

```
"""
```

```
def verify_credentials(creds_to_verify, cert_to_verify, target_urn, certificate,
                      credentials)
```

```
"""
```

Generates delegated slice credentials. The generated delegated credentials are not stored in the database.

Arguments:

delegatee\_cert: Certificate of delegatee  
 issuer\_key: Private key of the delegator  
 privileges\_list: List of privileges to be delegated. This must be a subset of privileges embedded in credentials.  
 expiration: Expiry date and time for delegated credentials  
 delegatable: if TRUE, delegated credentials can further be delegated.  
 issuer\_certificate: Certificate of delegator  
 slice\_credentials: Slice credentials owned by delegator and to be delegated

Return:

If successful SFA formatted delegated credentials

```
"""
```

```
def delegate_credentials(delegatee_cert, issuer_key, privileges_list, expiration,
                      delegatable, issuer_certificate, slice_credentials)
```

```
"""
```

Updates project & slice credentials after member certificate is regenerated due to his membership renewal or revocation.

When a member's certificate is renewed, all credentials generated with previous certificate become obsolete. With the help of this method, member certificate information in all relevant credentials can be updated. This method is not applicable for delegated credentials because they are not stored in SA database.

Arguments:

member\_urn: URN of the member whose certificate information has to be updated in slice and project credentials  
 certificate: Certificate of the method calling entity  
 credentials: Credentials of the method calling entity  
 options: A dictionary containing new member certificate

Return:

None if successful, otherwise error code.

"""

```
def update_credentials_for_member(member_urn, certificate, credentials, options)
```

## A.2 Member Authority (MA) API methods

"""

Verifies if given certificate is valid and Trusted by this C-BAS instance. It is intended for use with login mechanism, e.g., certificate based login in Expedient.

Arguments:

cert\_to\_verify: certificate to verify  
 certificate: Certificate of the method calling entity  
 credentials: Credentials of the method calling entity

Return:

Error message in case of failure.

"""

```
def verify_certificate(cert_to_verify, certificate, credentials)
```

"""

Returns an updated Certificate Revocation List (CRL) that must be used in conjunction with certificate and credential verification process.

Arguments:

certificate: Certificate of the method calling entity  
 credentials: Credentials of the method calling entity

Return:

CRL in PEM format digitally signed by MA

"""

```
def get_crl(certificate, credentials)
```

## B API method of FAPI Plugin for Expedient

|| || ||

Uses FAPI calls to creates a slice in C-BAS with the given name.

Arguments:

```
owner_urn: URN of slice owner
owner_certificate: Owner certificate required for slice credentials creation
slice_name: Desired name of slice
slice_desc: A short description of slice
slice_project_urn: URN of the project of which owner is member and wants to
                  create slice
```

Returns:

If successful returns URN of newly created slice.

|| || ||

```
def create_slice(owner_urn, owner_certificate, slice_name, slice_desc,
                slice_project_urn)
```

|| || ||

Uses FAPI calls to create a project in C-BAS with the given name.

Arguments:

```
certificate: Certificate of owner required for project credentials creation
credentials: System credentials of user to authorize the action
project_name: Desired name of the project
project_desc: A short description of the project
```

Returns:

If successful returns URN of newly created project.

|| || ||

```
def create_project(certificate, credentials, project_name, project_desc)
```

|| || ||

Adds a member to the project with default privileges

Arguments:

```
project_urn: URN of the project to which member is to add
to_add_user_urn: URN of the member
to_add_user_certificate: Certificate of the member
authz_user_urn: URN of the member who is requesting this action.
authz_user_certificate: Certificate of the member who requests this action
```

Returns:

None.

|| || ||

```
def add_member_to_project(project_urn, to_add_user_urn, to_add_user_certificate,  
                          authz user urn, authz user certificate)
```

```
"""
```

```
Removes a user's membership from a project.
```

```
Arguments:
```

```
    project_urn: URN of the project to which member should be added
    user_urn: URN of the member
    authz_user_urn: URN of the member who is requesting this action.
    authz_user_certificate: Certificate of the member who requests this action
```

```
Returns:
```

```
    None.
```

```
"""
```

```
def remove_member_from_project(project_urn, user_urn, authz_user_urn,
                               authz_user_certificate)
```

```
"""
```

```
Adds a member to the slice with default privileges
```

```
Arguments:
```

```
    project_urn: URN of the project to which this slice belongs
    slice_urn: URN of the slice
    to_add_user_urn: URN of the member
    to_add_user_certificate: Certificate of the member
    authz_user_urn: URN of the member who is requesting this action.
    authz_user_certificate: Certificate of the member who requests this action
```

```
Returns:
```

```
    None.
```

```
"""
```

```
def add_member_to_slice(project_urn, slice_urn, to_add_user_urn,
                        to_add_user_certificate, authz_user_urn,
                        authz_user_certificate)
```

```
"""
```

```
Removes a user's membership from a project.
```

```
Arguments:
```

```
    slice_urn: URN of the slice to which member should be added
    user_urn: URN of the member
    authz_user_urn: URN of the member who is requesting this action.
    authz_user_certificate: Certificate of the member who requests this action
```

```
Returns:
```

```
    None.
```

```
"""
```

```
def remove_member_from_slice(slice_urn, user_urn, authz_user_urn=None,
                              authz_user_certificate=None)
```

```
"""
```

Performs a lookup for slice credentials of a user.

Arguments:

project\_urn: URN of the project to which this slice belongs  
 slice\_urn: URN of the slice  
 user\_urn: URN of the member  
 user\_certificate: Certificate of the member

Returns:

If successful returns slice credentials.

"""

```
def get_slice_credentials(project_urn, slice_urn, user_urn, user_certificate)
```

"""

Performs a lookup for project credentials of a user.

Arguments:

project\_urn: URN of the project  
 user\_urn: URN of the member

Returns:

If successful returns project credentials.

"""

```
def get_project_credentials(project_urn, user_urn)
```

"""

Performs a lookup for member information based on username. If user does not already exist then registers it with given details

Arguments:

username: Username of the member  
 user\_details: User details required for registration, like username etc.

Returns:

If successful returns user details.

"""

```
def get_member_info(username, user_details=None)
```

"""

Renews system membership of a user through regeneration of user credentials and certificate

Arguments:

user\_urn: URN of the member  
 certificate: Certificate of the member who authorizes this action  
 credentials: Credentials of the member who authorizes this action

Returns:

If successful returns member certificate, certificate key, and credentials



```

"""
def regenerate_member_creds(user_urn, certificate=None, credentials=None)

"""
Generates a new SSH key pair and overwrites given member's public SSH key
with the new one.

Arguments:
    user_urn: URN of the member
    username: Username of the member
    certificate: Certificate of the member who authorizes this action
    credentials: Credentials of the member who authorizes this action

Returns:
    If successful returns public and private SSH key pair
"""
def regenerate_ssh_keys(user_urn, username, certificate=None, credentials=None)

"""
Updates a member's public SSH key in C-BAS database

Arguments:
    user_urn: URN of the member
    pub_ssh_key: New public SSH key of the member
    certificate: Certificate of the member who authorizes this action
    credentials: Credentials of the member who authorizes this action

Returns:
    True if successful otherwise False
"""
def update_ssh_key(user_urn, pub_ssh_key, certificate=None, credentials=None)

"""
Registers a new user with the C-BAS

Arguments:
    cert_str: Member certificate in string format
    certificate: Certificate of the member/entity who authorizes this action
    credentials: Credentials of the member/entity who authorizes this action

Returns:
    User details, like certificate, credentials, SSH keys.
"""
def register_user(username, user_details, certificate=None, credentials=None)

"""
Verify a member certificate through C-BAS

Arguments:

```

```
cert_str: Member certificate in string format
Returns:
    True if verification passes otherwise False
"""
def verify_certificate(cert_str)

"""
Checks the status of C-BAS instance by calling FAPI method 'get_version'

Arguments:
    None
Returns:
    True if C-BAS is reachable otherwise False
"""
def is_cbas_server_active()
```