Πανεπιστήμιο Θεσσαλίας,

Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

Μεταπτυχιακή εργασία

«Design and implementation of a resource discovery, reservation and provisioning framework for federated experimental facilities»

Σταυρόπουλος Δονάτος

UNIVERSITY OF
THESSALY

Επιβλέπων καθηγητής:

Κοράκης Αθανάσιος (Λέκτορας)

Συνεπιβλέπων καθηγητές:

Λέανδρος Τασιούλας (Καθηγητής)

Αργυρίου Αντώνιος (Λέκτορας)

Βόλος, Ιούνιος 2014

Ευχαριστίες

Με την εκπόνηση της παρούσας μεταπτυχιακής εργασίας, φέρνω εις πέρας τις μεταπτυχιακές μου σπουδές στο Τμήμα Μηχανικών Η/Υ Τηλεπικοινωνιών και Δικτύων του Πανεπιστημίου Θεσσαλίας.

Θα ήθελα αρχικά να ευχαριστήσω θερμά τον Λέκτορα του Τμήματος Μηχανικών Η/Υ Τηλεπικοινωνιών και Δικτύων κ. Κοράκη Αθανάσιο, και τον καθηγητή του Τμήματος Μηχανικών Η/Υ Τηλεπικοινωνιών και Δικτύων κ. Λέανδρο Τασιούλα, για τις χρήσιμες συμβουλές και υποδείξεις τους καθώς και για την υποστήριξη που μου προσφέρανε κατά τη διάρκεια της φοίτησής μου, αλλά και κατά την εκπόνηση της διπλωματικής μου εργασίας. Επιπροσθέτως να τους ευχαριστήσω για την δυνατότητα που μου δίνουν να βρίσκομαι στην ομάδα του NITlab και να ασχολούμαι συνεχώς με νέες τεχνολογίες και να κάνω έρευνα στον χώρο τον ασύρματων δικτύων.
Από καρδιάς να ευχαριστήσω όλη την ομάδα του NITlab μέσα από την οποία μαθαίνω διαρκώς καινούργια πράγματα και κάνω σημαντικές συνεργασίες.

Τέλος, ευχαριστώ θερμά την οικογένειά μου για την αμέριστη συμπαράσταση που μου παρείχε όλα αυτά τα χρόνια για την ολοκλήρωση των μεταπτυχιακών και προπτυχιακών σπουδών μου.

1 TABLE OF CONTENTS

ABSTRACT

In this work we present the design and implementation of a framework for resource discovery, reservation and provision in federated experimental facilities. We go through all the steps from the architectural design and implementation of the framework to its deployment on the testbed of NITOS. Initially a thorough analysis is given about the desired characteristics of the framework based on the requirements that stem from our experience of running NITOS testbed and those accumulated from our involvement in relevant research projects regarding the Future Internet (FI) experimental facilities. Finally we discuss how the interoperability of the framework with the NITOS local testbed services has been achieved.

More specifically we introduce the reader to the world of the FI experimentation and its various protocols that have been developed during the last years and form an integral part of the FI experimental facilities. We give an overview of the NITOS FI experimental facility and its control and management framework.

After providing the necessary background information to the reader, we move on by discussing the multilateral problem we are trying to solve from the perspective of a testbed administrator. We continue by providing a thorough analysis of the architectural components of the framework and its interconnections and relationships. More technical details follow in the section where the implementation specifics are given along with the corresponding explanation.

We conclude by showcasing the integration of the framework in the NITOS testbed. NITOS features its own resources and testbed-specific services, and the corresponding hooks needed to be implemented for a smooth integration. We envisage to further extend our framework with more capabilities and inspire other testbed owners to adopt our framework for facilitating their management of the facility.

2 INTRODUCTION

Research into the Internet of future has aroused a lot of interest lately, inducing the proliferation of experimental facilities which are also known as testbeds. Testbeds comprise programmable networking elements available to the experimenters who want to evaluate their algorithms and protocols in real world settings. Testbeds go a bit further from the constrained laboratory environment to large scale experimentation providing topologies that span the globe.

Towards the support of global scale experimentation several research projects have funded the federation of testbeds in the domain of interconnectivity as well as in the domain of the adoption of common tools and interfaces. Namely the European projects Openlab and Fed4FIRE as well as the GENI [1] project in the US are some of the most prominent endeavors in the area of Future Internet (FI) testbeds federation. During these projects, several decisions were made mainly towards the adoption of specific protocols regarding the management of the testbeds and the control of their resources.

In order to make large scale experimentation feasible, reproducible and easy, a resource control protocol was introduced called Federated Resource Control Protocol (FRCP) [2] so that to standardize the way experimenters interact with resources of the testbeds. This is a result of the aforementioned projects and its main purpose is to introduce a well-defined messaging protocol between user experiment tools and the components responsible for controlling the resources of the testbeds that are involved in the experiments.

Apart from the control of the resources, another protocol responsible for the management of the testbed has been introduced and was named Slice-based Federation Architecture (SFA) [3]. The main purpose of this protocol is to standardize the interface each testbed exposes through its main software component usually called Aggregate Manager (AM) where all the services of the testbed are concentrated. These services usually are responsible for testbed resource discovery, reservation and provision. By defining such an API the federation of testbeds becomes easier as all the services of the heterogeneous testbeds are exposed through a common interface which can be reached by SFA clients used by the experimenters.

Taking all the above into consideration, someone must be aware of these protocols when it comes to designing and implementing tools for management and administration of experimental facilities. The task of managing a testbed is not to be underestimated, since it involves several mechanisms related not only to AAA (Authentication, Authorization, Accountability) but also to resource advertisement, reservation and provisioning as well as policy enforcement. The ultimate objective of a testbed software management framework is to aid the administrators of experimental facilities in their everyday tasks of the management of the facilities.

The rest of this document is organized as follows. More details are given for the NITOS [4] FI facility, its control and management framework and the SFA protocol in the following subsections. Section 3 discusses the problem in more detail before moving on to section 4 and 5 where the design of the architecture and the implementation details are given respectively. In section 6 we evaluate the integration of the framework in the NITOS FI facility and in section 7 we conclude and outline directions for future work.

2.1 NITOS Future Internet Experimental Facility

The NITOS FI experimental facility operated by the University of Thessaly (UTH), forms one of the FIRE [5] infrastructures that is continuously evolving through major extensions that reflect the latest technologies and trends in the FI ecosystem. NITOS is currently capable of offering several testbed facilities to the interested communities that feature technologies in the domain of: wireless networks (Wi-Fi, WiMAX, and LTE), wired networks, opportunistic networks, Internet of Things (IoT), Software Defined Radios and Networks, energy consumption and cloud computing. In more detail, NITOS FI experimental facility comprises the following testbeds:

- Wi-Fi outdoor testbed
- Wi-Fi indoor testbed
- OpenFlow testbed
- Software Defined Radio testbed
- WiMAX testbed
- LTE testbed

Additionally, UTH is backed-up with a cloud infrastructure capable of serving all the needs of computation power and storage capacity, acting complementary to the FIRE infrastructure described above. The cloud infrastructure helps the experimenters to scale and stretch their experiments when the normal computing power and storage of the nodes is not sufficient. The overall architecture of NITOS can be seen in Figure 1 below.

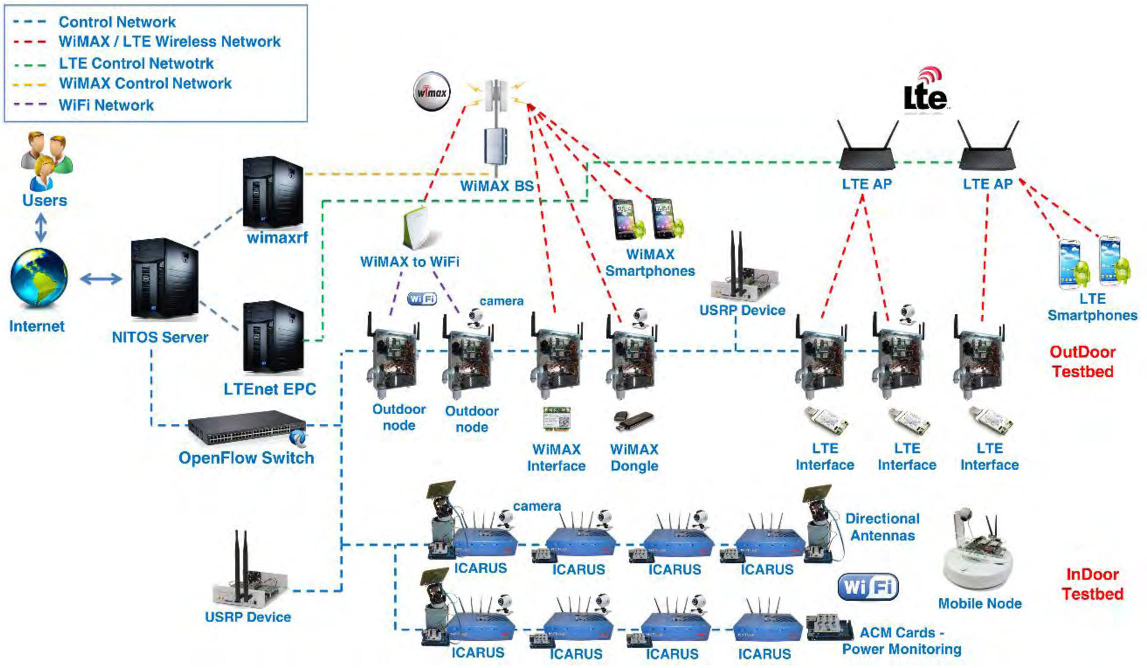


Figure 1: NITOS Testbed Architecture

2.2 cONTROL AND MANAGEMENT FRAMEWORK (OMF)

The control and management of the NITOS facility is being done using the OMF [6] open-source software. OMF was originally created in the Orbit [7] testbed, and soon became the prevalent tool for experimentation in networking testbeds providing a modular architecture capable of controlling heterogeneous resources. Notably, OMF is the primary experiment control tool in the FIRE [5] initiative as well as in GENI [1].

The objective of OMF is to provide a more standard and flexible way in controlling testbed resources in contrast to the user's custom scripts who are usually tailor-made according to a specific testbed where the experiment is conducted. In particular, OMF enables the experimenter to automate an experiment instead of setting up everything manually by logging into each node to configure/control its operation. The concept is similar to network simulators where the user describes a topology along with the applications that run during the simulation. The difference is that the topology consists of physical nodes on which OMF runs applications like a traffic generator. The configuration and control of node operation occurs through specific properties, which are part of “formal” resource descriptions, and can be done not only at experiment setup but also during experiment runtime.

The basic components of the OMF framework are the Experiment Controller (EC) and the Resource Controllers (RCs). The role of the EC is to orchestrate the execution of the experiments, written in the OMF Experiment Description Language (OEDL). The EC interprets OEDL and sends appropriate messages to the corresponding RCs. In turn, each

RC is responsible for abstracting and controlling one or more underlying physical or logical resources. It basically converts the messages received from the EC into resource-specific commands, and relays the response back to the EC. It is important to note that the message exchange between the EC and the RCs is performed using publish-subscribe mechanism and the exchanged messages are basically the FRCP messages. FRCP is actually a publish-subscribe protocol agnostic to the pubsub technology used and currently there are two different pubsub technologies used in the OMF. The first option uses FRCP over XMPP pubsub communication whereas a second option was added recently through the usage of FRCP over AMQP pubsub protocol. An overview of the OMF architecture and its components can be seen in Figure 2.

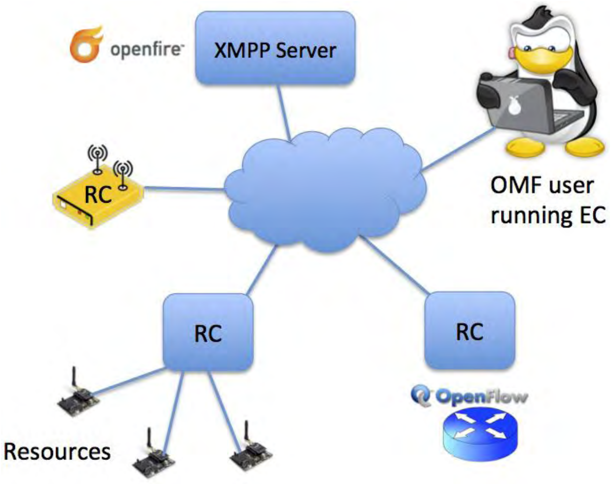


Figure 2: Overview of the OMF architecture

2.3 SLICE-BASED FEDERATION ARCHITECTURE (SFA)

SFA [3] forms the control plane for discovering and allocating resources offered by a federation of networking testbeds. It was first introduced in the context of federating PlanetLab Central and PlanetLab Europe as part of the EU's OneLab and OneLab2 projects. Since then, SFA has been used in GENI and FIRE as the adopted control plane architecture and relevant implementations appeared in this context. Currently these implementations differ in the details but the main idea remains the same. Efforts are being made such that all involved parties will converge to a single SFA version with no differences at all.

SFA has been designed to provide a minimal set of functionalities that a testbed can implement in order to enter into a global and interoperable federation. SFA introduces a fully distributed solution in which each testbed is the responsible entity for the resources that it brings, and each user community, along with its experiments, is represented by an authority. An experimenter in an SFA-based environment can transparently browse resources on any federated testbed, and allocate and reserve those resources. SFA's main functionalities are the Authentication of users through certificates issued by respective trusted authorities, resource advertisement such that the users can browse the available resources and finally resource allocation/provision of the desired resources to the users/experimenters.

3 PROBLEM STATEMENT

The heterogeneity of the resources that often constitute a testbed, imposes a significant problem for the management and the administration of the facility. Complex resources necessitate the analogous complex adaptations on the side of the management tools. Administrators should be able to provide a proper resource advertisement to the experimenters through their testbed tools, containing all the available resources along with their specific details. To accomplish this, a testbed management tool initially should feature an inventory service capable of storing and describing the unique characteristics of each resource of the facility and at the same time should be modular and extensible in order to include possible newly added resources without the need of modifying other parts of the testbed software components.

Provided that an experimenter has picked the resources for his experiment during the resource discovery phase described above, the next step will be to have these resources provisioned and ready for his experiment. This task should be performed by the management framework of the testbed and involves the initialization of possibly different types of resources, meaning that for every different type of resource there should be implemented a different routine for instantiating that resource. For instance, the instantiation of a Virtual Machine (VM) differs a lot from the booting of a physical machine or the initial configuration of a WiMAX base station. All this diversity in the nature of the resources creates extra problems that require the attention and resolution from the testbed operators so that to ensure automation and interoperability in their testbed functionalities.

Moreover, features like resource reservation and policy enforcement is of great interest to the administrators of experimental facilities who seek ways to further utilize their platforms through fine-grained policies. A common problem for the administrators is the enforcement of resource usage quotas based on the user role which might be a student, an academic researcher or an industrial researcher. Additionally, the participation of the testbeds in federations introduces new burdens to the administrators who want to apply different policies on users based on their corresponding administrative domains.

Besides the local communication interfaces used between the testbed's software components providing flexibility in the testbed management, further versatility is needed in terms of communication interfaces and interoperability with external or 3rd party tools and even with other testbeds which run under a different administrative domain. The latter feature is about federating with other testbeds, which is tackled by the SFA protocol. This adds an extra requirement to the administrators of the experimental facilities who want to federate their testbed with other testbeds, since it implies developing the needed adaptation layers between their management software and the SFA interface which will be used for federating.

4 ARCHITECTURE

During the design phase of our framework, the numerous problems/functionalities that were mentioned in the previous section, were taken into consideration in the scope of implementing an integrated solution for testbed management. The expected functionality of such a framework and its desired characteristics can be summarized in the following:

- Authentication, Authorization and Accounting (AAA) mechanisms
- Service Oriented Architecture (SOA) in order to provide its functionality as a service to other applications or components of the testbed.
- Multiple communication interfaces supporting widely used protocols in the field of testbeds like the aforementioned SFA and FRCP.
- Capability for in-advance reservation of resources.
- Policy enforcement along with resource usage quotas.

In the following paragraphs a more in depth analysis will be provided about the various components of the implemented framework and its corresponding functionalities. The name of our framework is “Broker” as it includes functionality which can be correlated to that of the brokering service between experimenters and testbeds. We will start by describing the architecture from the outer to the inner by beginning with the available communication interfaces which can be seen in Figure 3.

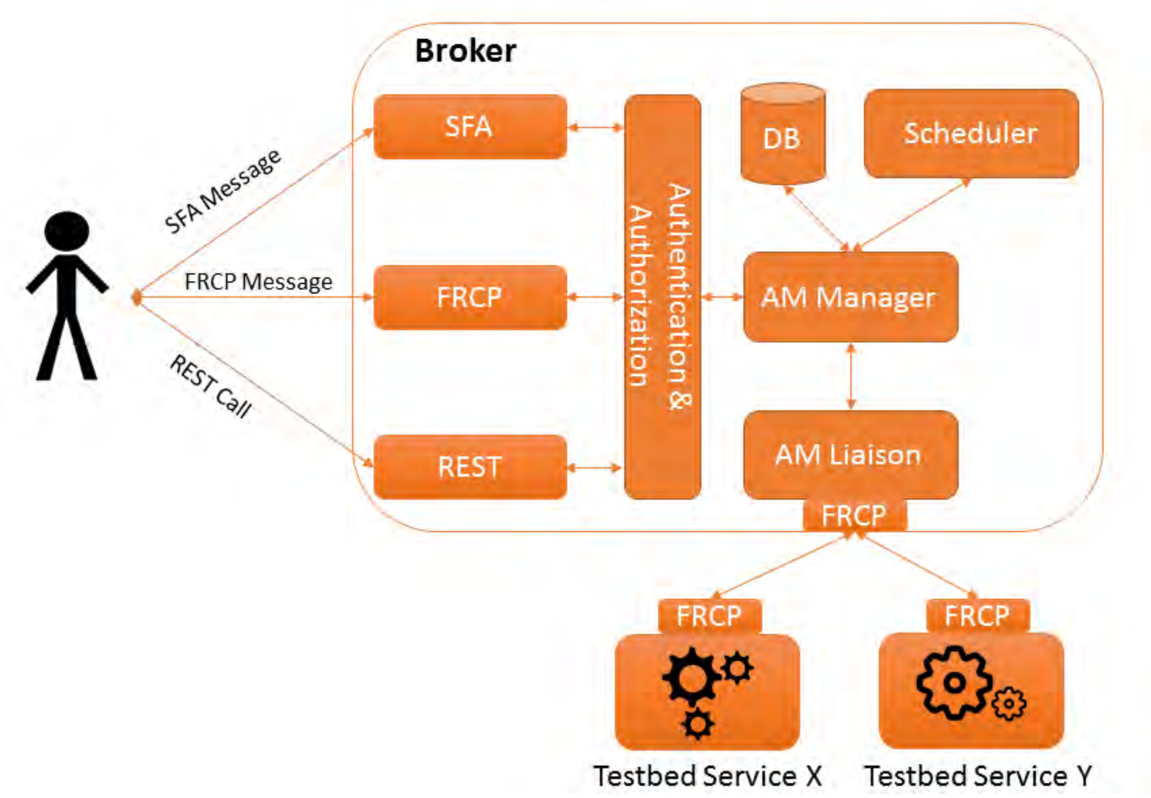


Figure 3: Broker Architecture

4.1 COMMUNICATION INTERFACES

The Broker comprises three different communication interfaces, namely an SFA API, an FRCP API and a REST API. The SFA is based on the XML-RPC technology, whereas the FRCP API is currently using XMPP as the underlying transport protocol. The REST API as usual runs over the HTTP protocol and uses JSON for data representation.

Regarding the SFA API, currently the Broker features an implementation of the GENI SFA AM API v2 defined in [8] and through this API the Broker is able to advertise the resources stored in its inventory, accept reservations of the resources as well as requests for provisioning the reserved resources. In principal, this interface will be used by SFA clients like the “SFI”, “OMNI” and the graphical user client tool “MySlice” which has the form of a portal, facilitating that way the inexperienced users.

Secondly the FRCP API is chiefly used in the communication with the resources of the testbed through the corresponding API of their resource controllers. This API can also be seen as the southbound of the Broker where it contacts the different components of the testbed in order to configure them or even trigger an action necessary for the provisioning stage.

The third communication interface is a REST API which provides a point where all the functionalities of the Broker can be reached as a web service either by a client tool or a testbed tool. For instance, the REST API is used for implementing a web interface for the users of the testbed where they can see the available resources and reserve some of them for their experiments. Given that the Broker can be reached through the REST interface, this leaves us with the flexibility to host the web interface to a different location from the Broker and their interoperability is performed over the REST calls.

4.2 AUTHENTICATION

The next layer after a message is received in one of the communication interfaces, is the Authentication layer which is responsible for determining if the other point can be authenticated or not. The chosen authentication mechanism in SFA uses a Public Key Infrastructure (PKI) based on X.509 certificates. Each user can obtain a valid certificate from the authority of its administration domain where he belongs and use that in an SFA client in order to contact the various testbeds of a federation through the SFA API. In this context, we have implemented a mechanism in the authentication layer which parses and verifies the users' certificates against some trusted root certificates which usually are the certificates of the trusted authorities in a federation.

Similarly, the authentication layer in the REST API uses the X.509 certificates in order to accept or reject the incoming messages. We have chosen to base the authentication in the REST API on the usage of the same certificates as in the SFA in order to harmonize the way we authenticate the users. Thus, no further actions are needed to be made from the users for having themselves authenticated in the various APIs as they can use a single certificate obtained from their authority.

The FRCP API uses X.509 certificates in order to authenticate the exchanging messages between the entities but currently they have slight differences in their attributes with those used in the SFA domain. However, the main idea remains the same and their contents are about to converge soon for facilitating the experimenters.

4.3 AUTHORIZATION

The next layer a message comes through is that of the Authorization layer. Having authenticated the user, now we would like to check if the user is authorized to perform the actions he asks for. In contrast to the authentication which is basically the same mechanism all over the three different APIs, the authorization is different in each one of them. For example, in the SFA a signed XML carries the credentials of the user, whereas in FRCP signed assertions at the end of the messages are enabling the authorization of the users. Last in the REST we base our authorization decisions on the user id obtained from its certificate.

The authorization layer gives the ability to the testbed owner to define the desired policies based on the authenticated user and its credentials. A more technical overview of the authorization component will be given in the next section where the testbed administrator can modify according to his needs.

4.4 Inventory management & scheduling

In the inner blocks of the Broker we distinguish two main entities, the “manager” and the “scheduler”. The first entity has the role of the orchestrator as all the core functionalities regarding the inventory are implemented in this component. In essence, the “manager” is responsible for modifying the inventory by adding/removing resources from it as well as for seeking resources based on the queries it receives from the rest of the components. The “manager” is tightly connected with the authorization layer as it consults it in order to decide whereas an action is allowed to be performed or is forbidden.

The “scheduler” entity is responsible for allocating resources to experimenters in the time domain after receiving corresponding requests from them. The “scheduler” is the point where all the allocation policies take place, simple or complex. The simplest policy can be the Best Effort logic, meaning that, simply checks the availability of the resources in the given time period and if they are available, it reserves them on behalf of slice/account. Another more complex example will be that a user specifies some preferences, like minimum duration, preferable duration, minimum resources, preferable resources and the scheduler tries to do the best it can, to fulfill these preferences. It can also use the policy information to decide which user must have the resources, whenever there are conflicts between the users’ requests.

4.5 Interoperability with testbed services

The last component of the Broker is called “AM Liaison” and it is responsible to connect the Broker with the testbeds internal services whether these are aggregated in a manager or distributed in various software components. The “AM Liaison” leverages the FRCP API in order to communicate with the underlying components, as long as they are FRCP enabled. For instance, when the Broker receives a request for a Virtual Machine provision, the “AM Liaison” is the responsible entity to contact through FRCP the entity capable of provisioning the requested VM. By keeping these testbed-specific tasks outside of the Broker architecture we are able to provide a more generic framework for easier adoption from heterogeneous testbeds requiring minimal modifications.

5 SYSTEM IMPLEMENTATION

In this section a more technical description will be given regarding the Broker's components and their specific implementation. To start with, the Broker was implemented using the "Ruby" programming language for several reasons. Firstly, "Ruby" as most of the "scripting" languages provides an optimal tradeoff between ease of implementation, scalability and performance since our implementation is quite complex but at the same time not time critical as a video codec for instance. Secondly, the OMF framework is implemented in Ruby and we took advantage of the FRCP implementation by including and using it for the FRCP API of the Broker. Last we leveraged Ruby's metaprogramming capabilities in order to implement a working inventory which can be extended based on each testbed owner preferences and specific resources without the need of modifying the core functionalities.

Regarding the library used in the implementation of the inventory was the "Datamapper" which is an Object-Relational Mapper (ORM) variant implemented in Ruby. This library enables us to define a basic information model solely by naming the Classes along with their attributes and their inheritance without the need to worry about the underlying SQL statements that are automatically generated by the "Datamapper". The concept of our inventory was to define a basic resource set that others could build upon it. Thus we defined a class named Resource with the following basic attributes that every resource should have:

- ID (a unique id)
- Type (What is the type of this resource, e.g. Node)
- UUID (Universal Unique Identifier)
- Name
- URN (Uniform Resource Name, "urn:publicid:IDN+domain+type+name")

The next basic resource we defined is the Component which inherits from the Resource all the attributes and features some more:

- Domain (The administrative domain in which this resource belongs, used in the URN)
- Exclusive (Denotes if this resource is an exclusive one or can be shared through virtualization methods)
- Available (Resource availability)
- Status (Operation status of the resource)

The complete information model along with the relationships and the attributes of the components can be seen in Figure 4. So if someone is willing to include a new resource in the inventory all he has to do is to describe it in its own class by inheriting one of the existing classes. Then the new type of the resource will be automatically exposed by the Broker's API and will also be modified through the aforementioned APIs.

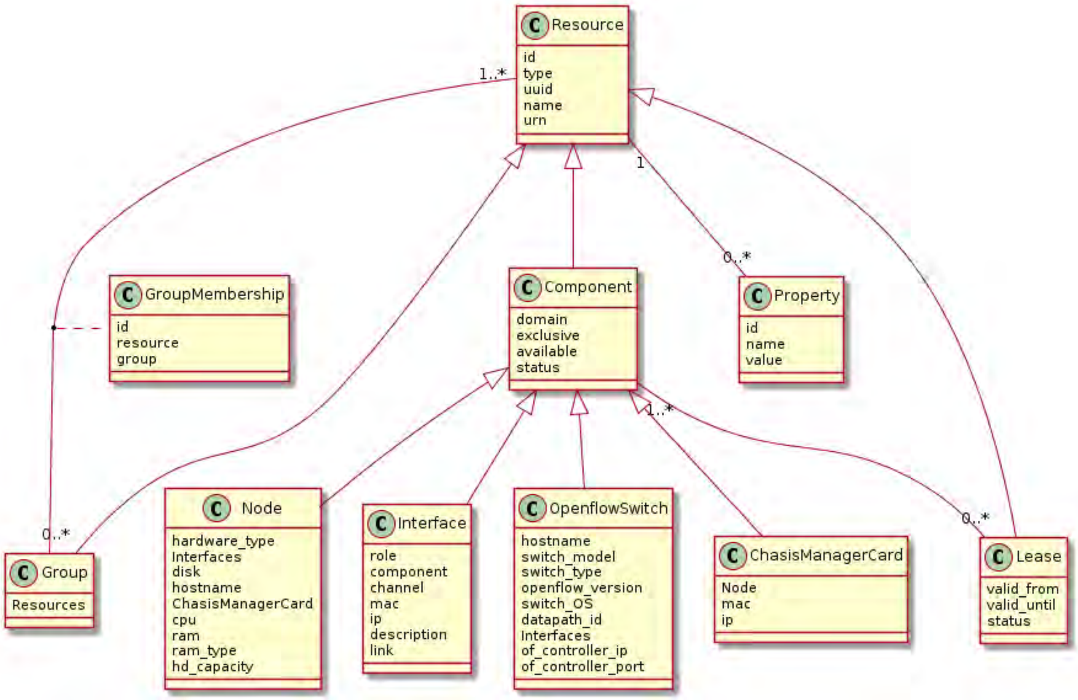
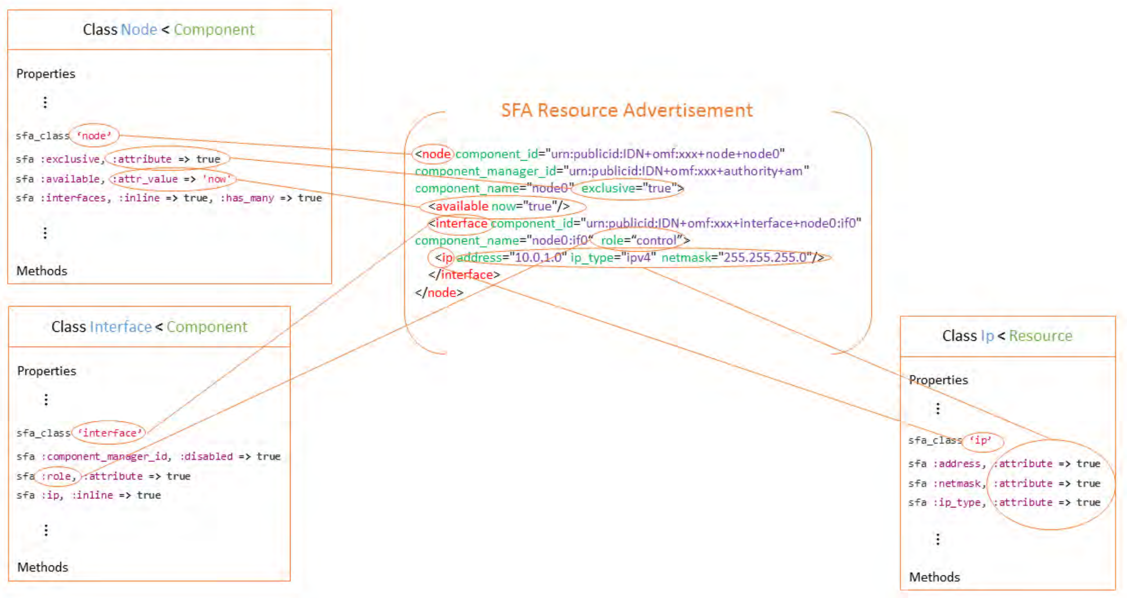


Figure 4: Information Model

Provided that we wanted also the resources to be exposed through SFA with a specific manner using XML serialization and thus forming the RSpec file that is called in SFA the advertisement of the resources, we implemented an annotation mechanism which allows to decide on how the resources and their attributes are going to be serialized. An example is given in Figure 5 with a node where we choose which attributes we want to expose through SFA and what form we want the XML nodes to have.



```
Class Node < Component

Properties
    ⋮
sfa_class 'node'
sfa :exclusive, :attribute => true
sfa :available, :attr_value => 'now'
sfa :interfaces, :inline => true, :has_many => true
    ⋮
Methods
```

```
<node component_id="urn:publicid:IDN+omf:xxx+node+node0"
component_manager_id="urn:publicid:IDN+omf:xxx+authority+am"
component_name="node0" exclusive="true">
 <available now="true"/>
  <interface component_id="urn:publicid:IDN+omf:xxx+interface+node0:if0"
component_name="node0:if0" role="control">
   <ip address="10.0.1.0" ip_type="ipv4" netmask="255.255.255.0"/>
  </interface>
</node>
```

```
Class Interface < Component

Properties
    ⋮
sfa_class 'interface'
sfa :component_manager_id, :disabled => true
sfa :role, :attribute => true
sfa :ip, :inline => true
    ⋮
Methods
```

```
Class Ip < Resource

Properties
    ⋮
sfa_class 'ip'
sfa :address, :attribute => true
sfa :netmask, :attribute => true
sfa :ip_type, :attribute => true
    ⋮
Methods
```

Figure 5: Information Model to SFA annotations

6 INTEROPERABILITY – USE CASES

In this section we demonstrate how the Broker has been integrated in the NITOS facility and its several specific software components. In particular, we discuss how the Broker is able to fit in any testbed environment, given that the necessary hooks are implemented between the Broker's "AM Liaison" component and the other testbed services we would like to control. In the NITOS case we managed to connect the Broker with several testbed services including the remote power management of the nodes, the process of image burning to the nodes, the user handling through the creation/deletion of Linux accounts and the management of the OpenFlow switches.

6.1 POWER MANAGEMENT OF THE NODES

NITOS consists of physical machines which can be remotely managed by chassis manager (CM) cards which are responsible for powering on/off or resetting the physical nodes by providing an interrupt signal to the corresponding pins of the motherboard. The control of the CM cards is being done through a CMC service component which exposes an FRCP API and translates FRCP messages to commands sent to the CM cards. In this context, we wanted to authorize requests to this service based on the user identity and its resource reservations. To this end we leveraged the Broker's FRCP API in order to query the resource reservations of a user and to accept or reject his request for node operations. A more detailed sequence diagram can be seen in Figure 6.

In this diagram, an experimenter makes a request through OMF to switch on a specific Node. The messages exchanged are mainly between the resource controller of the CM cards and the Broker. Initially a configure message arrives from the experimenter containing all the necessary information like the id of the node and the desirable state (on/off/reset) of the node. The CM controller interacts with the Broker's FRCP API in order to request the available nodes and the leases of the experimenter. After having all these information determines if the experimenter has reserved the node before proceeding to the actual Node operation (on/off/reset). At the end, the CM controller informs the experimenter about the successful outcome of the command.

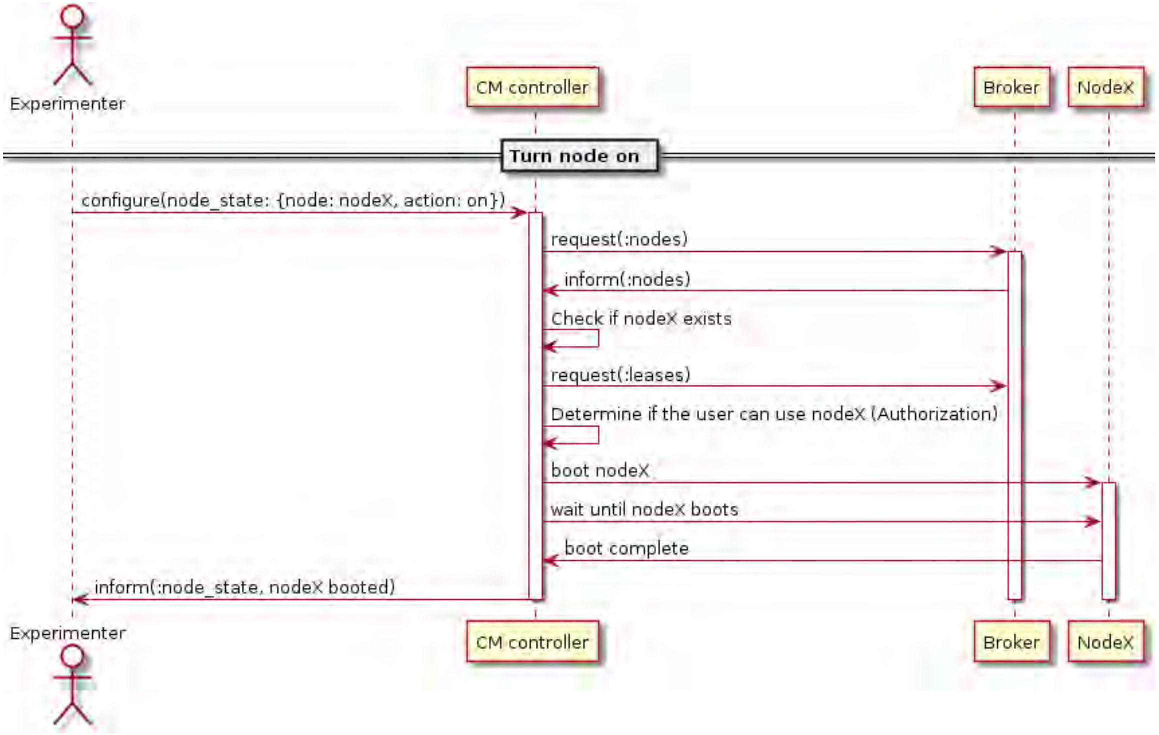


Figure 6: Node Operations

6.2 OS IMAGE BURNING SERVICE

Besides the service responsible for controlling the operation of the nodes, there is an additional service which loads or saves OS images from/to the hard disks of the nodes. Obviously this service needs to make similar decisions with that of the CMC service and thus it contacts the Broker every time a request is received from a user. Similarly it exposes an FRCP API and communicates with the Broker over this. A sequence diagram which shows a user request for loading an image is shown below in Figure 7.

In this diagram we distinguish several entities like the experimenter, the Broker, a Node and three Frisbee [9] relevant resource controllers. Frisbee is the service responsible for burning an OS image to one or several nodes simultaneously. The user initially requests through OMF his desirable OS image and the nodes he wants to be loaded. The resource

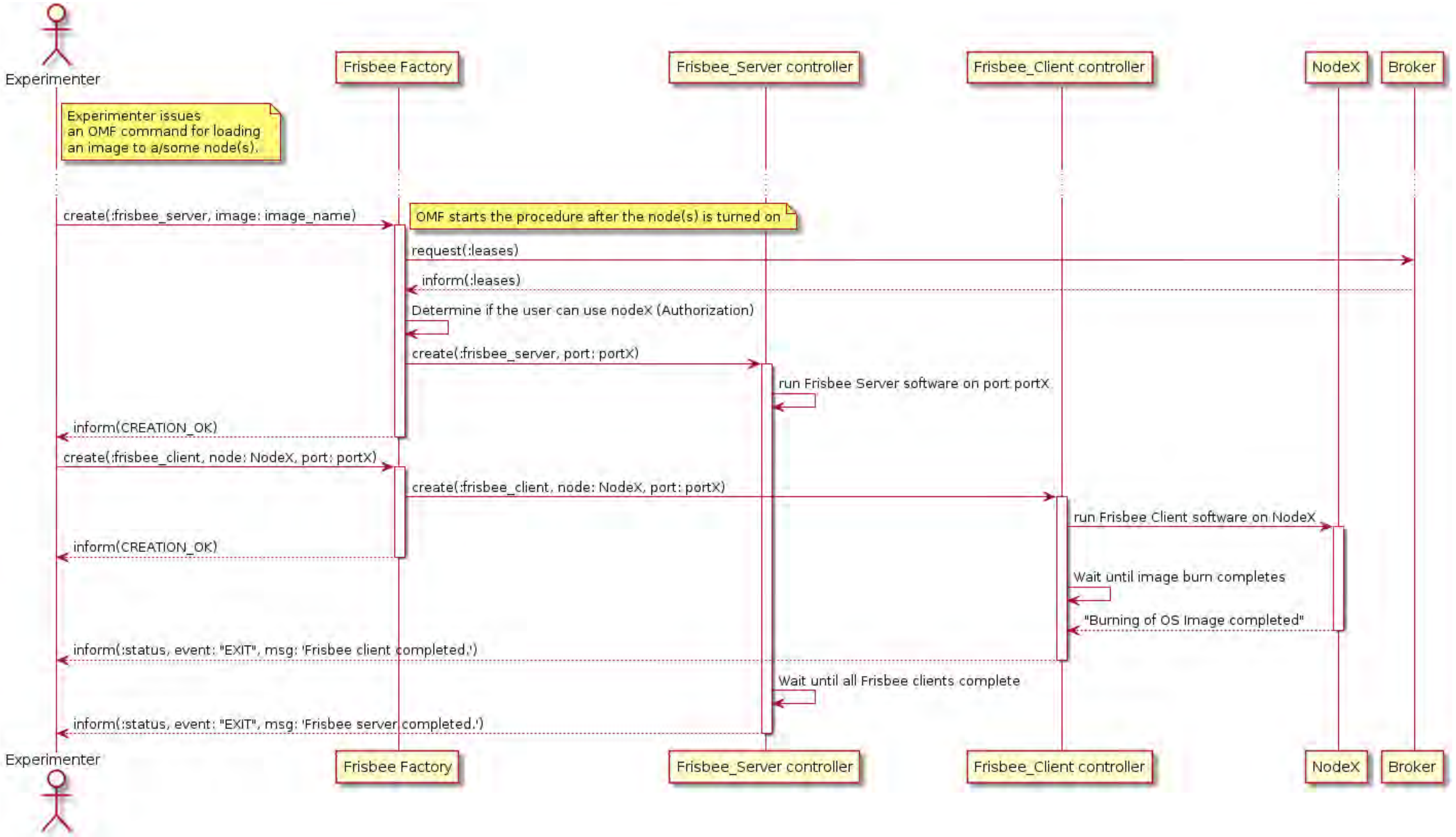

Experimenter
Frisbee Factory
Frisbee_Server controller
Frisbee_Client controller
NodeX
Broker
Experimenter issues
an OMF command for loading
an image to a/some node(s).
create(:frisbee_server, image: image_name)
OMF starts the procedure after the node(s) is turned on
request(:leases)
inform(:leases)
Determine if the user can use nodeX (Authorization)
create(:frisbee_server, port: portX)
run Frisbee Server software on port portX
inform(CREATION_OK)
create(:frisbee_client, node: NodeX, port: portX)
create(:frisbee_client, node: NodeX, port: portX)
run Frisbee Client software on NodeX
inform(CREATION_OK)
Wait until image burn completes
"Burning of OS Image completed"
inform(:status, event: "EXIT", msg: 'Frisbee client completed.')
Wait until all Frisbee clients complete
inform(:status, event: "EXIT", msg: 'Frisbee server completed.')
Experimenter
Frisbee Factory
Frisbee_Server controller
Frisbee_Client controller
NodeX
Broker

Figure 7: Image Burning Process

controller “Frisbee Factory” after making the necessary authorization checks with the Broker in order to determine whether the user has reserved the specified nodes or not, it continues by creating a “Frisbee server” resource controller. This starts an instance of a Frisbee server which listens for client connections. After that, the “Frisbee Factory” receives a create message from the experimenter’s OMF tool which asks to create the necessary Frisbee clients in order to connect to the server and start loading the image. The “Frisbee Client” resource controllers are responsible for running Frisbee clients on the Nodes and wait until the procedure is done before “killing” the running clients. The procedure terminates after all the clients have successfully completed their jobs. The experimenter receives a message that the loading of the image has been successfully ended.

6.3 User management service

Among the testbed-specific services of NITOS there is also the user management which is about Linux user account handling. Whenever a user acquires a slice in NITOS he is given a Linux account with whom it can login onto the testbed server and access the reserved resources. This service like the previous services features an FRCP interface and is responsible for the management of the users’ Linux accounts. For instance, when an SFA request is sent to the Broker asking for account creation (Create Sliver in the SFA language), the Broker contacts the User Management service and asks to create this account on behalf of the user who requested. A sequence diagram depicting this procedure is shown in Figure 8.

More specifically, the experimenter in the diagram sends an SFA request for creating a sliver in the testbed by providing his credentials. Given the Broker approves his request, it initiates the procedure of creating a Linux account for this slice. The Broker contacts the “User Factory” resource controller in order to create a “User” resource controller responsible for the actual creation of the Linux account on the server of the testbed. Besides the user creation, the User resource controller generates SSH keys if not there already and asks for a certificate to be signed by the Broker in order to be used when contacting local testbed services like the aforementioned image burning service and the node power management service. Last the Broker requests from the User resource controller to put the SSH public keys it received with the SFA request in the “authorized_keys” file, in order to grant remote access to the experimenter. At the end it releases the corresponding User resource controller as it is not needed anymore to modify the created Linux account.

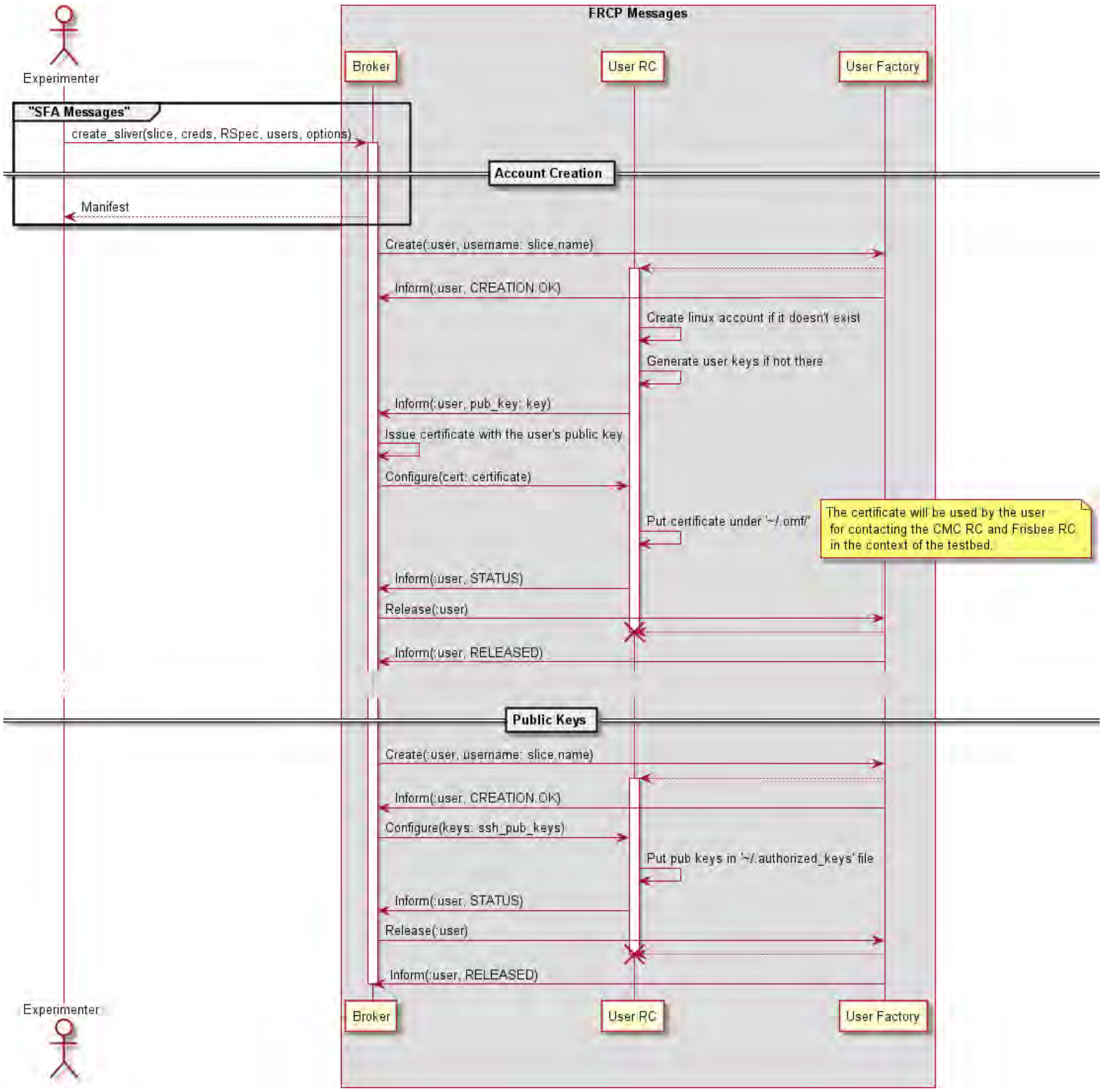

FRCP Messages
Experimenter
Broker
User RC
User Factory
"SFA Messages"
create_sliver(slice, creds, RSpec, users, options)
Account Creation
Manifest
Create(:user, username: slice.name)
Inform(:user, CREATION.OK)
Create linux account if it doesn't exist
Generate user keys if not there
Inform(:user, pub_key: key)
Issue certificate with the user's public key
Configure(cert: certificate)
Put certificate under '~/.omf/'
The certificate will be used by the user for contacting the CMC RC and Frisbee RC in the context of the testbed.
Inform(:user, STATUS)
Release(:user)
Inform(:user, RELEASED)
Public Keys
Create(:user, username: slice.name)
Inform(:user, CREATION.OK)
Configure(keys: ssh_pub_keys)
Put pub keys in '~/.authorized_keys' file
Inform(:user, STATUS)
Release(:user)
Inform(:user, RELEASED)
Experimenter
Broker
User RC
User Factory

Figure 8: User/Account Creation

6.4 OpenFlow Management service

Continuing on the identification of the various services hosted by the NITOS testbed, we move on to one more which is responsible for OpenFlow switch management. During the reservation process of the nodes of NITOS, users automatically reserve the corresponding ports (which the nodes are connected to) of the OpenFlow switches and thus his nodes are part of an isolated flow space (think of it as being connected in smaller switch which has only his nodes connected to it). The creation of this flow space is being done from the software component which is called FlowVisor. FlowVisor does not expose an FRCP API but a relevant Resource Controller (RC) has been developed [10] in NITOS for controlling the FlowVisor through FRCP commands. Given that, when a reservation occurs, the

Broker contacts the OpenFlow RC in order to create the necessary flowspace through the FlowVisor. A detailed sequence diagram can be seen below in Figure 9.

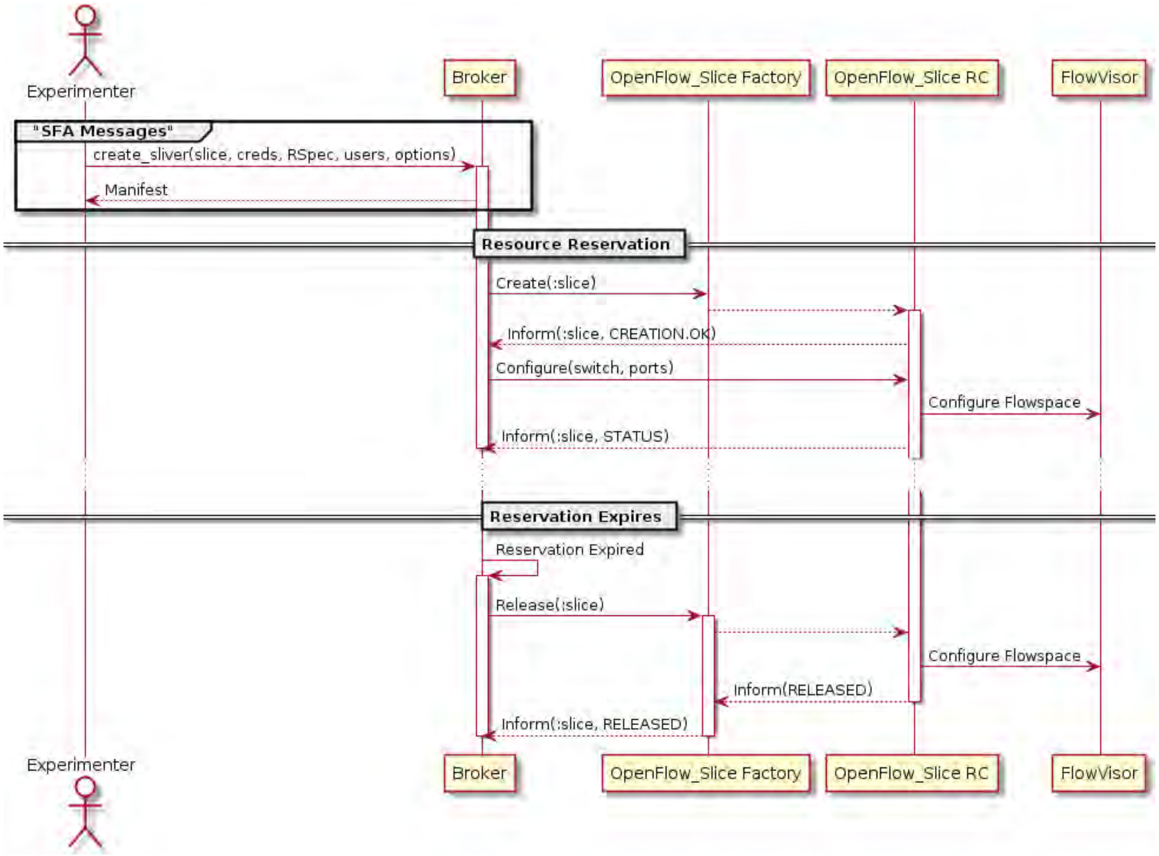


Figure 9: OpenFlow Slice Creation Procedure

In the diagram we can see the interactions between the Broker and the OpenFlow resource controllers when an SFA request arrives asking for resource reservation. The Broker initially requests from the OpenFlow Slice Factory RC to create an OpenFlow Slice RC which will be responsible for modifying the FlowVisor in order to create the flow space for this slice. After the expiration time of the reservation, the OpenFlow Slice RC contacts again the FlowVisor in order to remove the corresponding flow entries for the slice that its reservation has just expired. In both occasions, the resource reservation initiation and expiration the Broker is the entity who triggers the procedure for modifying the flow space through the FlowVisor.

7 Conclusion and Future Work

In this work we presented a framework for resource discovery, reservation and provisioning, designed for federated experimental facilities. We went through an analysis of the current status in the area of Future Internet (FI) experimentation using experimental facilities which are also known as testbeds and presented the requirements or characteristics a framework such that should feature. After clearly defining the problems that testbed framework should cope with we presented the whole architecture of the framework and described each component individually and the purpose it serves in the overall big picture. After that, we went through more technical details regarding the implementation choices we made about the framework and its components. In the last section we showcase the usage of the framework in the NITOS testbed and the way we achieved interoperability with the various testbed services which are responsible for different functionalities each one of them.

For future work, we envisage to further enhance our framework and keep it up to date with the latest developments in the area of FI facilities. For the beginning, we aim to upgrade our current SFA API from GENI AM v2 to v3 and continue its upgrade as soon as new versions of the SFA API are coming out. We are also targeting to extend our current information model with support for Ontologies so that to take advantage on the features the semantic web tools provide such as reasoning support over queries for resource discovery. Instead of using a simple XML schema for advertising the resources, the adoption of a common ontology among the FI testbeds will provide the necessary compatibility that will allow all the testbeds to advertise their resources based on the commonly adopted ontology. Last, we are planning to extend the current scheduler entity such that more scheduling and conflict resolution algorithms will be supported and a more sophisticated quotas enforcement mechanism will be supported.

8 REFERENCES

[1] M. Berman, J. S. Chase, L. Landweber, A. Nakao, M. Ott, D. Raychaudhuri, R. Ricci and I. Seskar, "GENI: A federated testbed for innovative network experiments," *Computer Networks, Elsevier,* 2014.

[2] "Federated Resource Control Protocol Specification," [Online]. Available: https://github.com/mytestbed/specification/blob/master/FRCP.md.

[3] "Slice-based Federation Architecture Specification," [Online]. Available: http://open-multinet.info/images/documents/open-multinet-sfa.html.

[4] "Network Implementation Testbed using Open Source platforms," [Online]. Available: http://nitlab.inf.uth.gr/NITlab/index.php/testbed/nitos-facility-architecture.

[5] "FIRE: Future Internet Research and Experimentation} initiative official Website," [Online]. Available: http://www.ict-fire.eu/.

[6] T. Rakotoarivelo, G. Jourjon and M. Ott, "Designing and Orchestrating Reproducible Experiments on Federated Networking Testbeds," *Computer Networks, Elsevier,* 2014.

[7] D. Raychaudhuri, I. Seskar, M. Ott, S. Ganu, K. Ramachandran, H. Kremo, R. Siracusa, H. Liu and M. Singh, "Overview of the ORBIT radio grid testbed for evaluation of next-generation wireless network protocols," in *3rd IEEE Conf on Wireless Communications and Networking (WCNC)*, 2005.

[8] "GENI Aggregate Manager API Version 2," [Online]. Available: http://groups.geni.net/geni/wiki/GAPI_AM_API_V2.

[9] M. Hibler and et al., "Fast, Scalable Disk Imaging with Frisbee," in *USENIX Annual Technical Conference, General Track*, 2003.

[10] "OMF Resource Controller for OpenFlow," [Online]. Available: https://github.com/kohoumas/omf_rc_openflow.