ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ Η/Υ, ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ ΚΑΙ ΔΙΚΤΥΩΝ

Σχεδιασμός και υλοποίηση υποδομής διαχείρισης για την διασύνδεση ετερογενών πειραματικών διατάξεων

Μεταπτυχιακή εργασία

του

Χαρίλαου Νιαβή

Επιβλέποντες: Λέανδρος Τασιούλας (Καθηγητής)

Σπύρος Λάλης (Αναπληρωτής Καθηγητής)

Δημήτριος Κατσαρός (Λέκτορας)

Βόλος, Φεβρουάριος 2013

Ευχαριστίες

Η παρούσα μεταπτυχιακή εργασία, σηματοδοτεί την ολοκλήρωση των μεταπτυχιακών μου σπουδών στο Τμήμα Μηχανικών Η/Υ Τηλεπικοινωνιών και Δικτύων του Πανεπιστημίου Θεσσαλίας.

Θα ήθελα αρχικά να ευχαριστήσω θερμά τον κ. Κοράκη Αθανάσιο, Λέκτορα του Τμήματος Μηχανικών Η/Υ Τηλεπικοινωνιών και Δικτύων, για τις χρήσιμες συμβουλές και υποδείξεις του καθώς και για την υποστήριξη που μου προσέφερε κατά τη διάρκεια της φοίτησής μου, αλλά και κατά την εκπόνηση της μεταπτυχιακής μου εργασίας.

Ευχαριστώ επίσης τον επιβλέποντα της εργασίας μου, Καθηγητή του Τμήματος Μηχανικών Η/Υ Τηλεπικοινωνιών και Δικτύων, κ. Λέανδρο Τασιούλα και τους συνεπιβλέποντες: τον Αναπληρωτή Καθηγητή του Τμήματος Μηχανικών Η/Υ Τηλεπικοινωνιών και Δικτύων, κ. Σπύρο Λάλη και τον Λέκτορα του Τμήματος Μηχανικών Η/Υ Τηλεπικοινωνιών και Δικτύων κ. Δημήτρη Κατσαρό για την καθοδήγησή τους.

Επίσης, θα ήθελα να ευχαριστήσω τον συνεργάτη μου Ιωάννη Ηγούμενο και τον μεταπτυχιακό φοιτητή του Τμήματος Μηχανικών Η/Υ Τηλεπικοινωνιών και Δικτύων Βασίλη Μαγλογιάννη, για την πολύτιμη συνδρομή τους στο να φέρω σε πέρας την μεταπτυχιακή μου εργασία.

Τέλος, ευχαριστώ θερμά την οικογένειά μου για την αμέριστη συμπαράσταση που μου παρείχε για την ολοκλήρωση των μεταπτυχιακών μου σπουδών.

Στην οικογένειά μου

CONTENTS

ABSTRACT

The present Final Project Dissertation – Thesis is the description of the process of making NITOS testbed, part of the federation community.

Our basic goal is to achieve the federation of NITOS testbed with other testbeds. Towards this goal, we decided to use the Generic SFA Wrapper, which is an implementation of the Slice – based Facility Architecture (SFA).

In order to accomplish the federation of NITOS testbed through the Generic SFA Wrapper, we had to implement these three basic tasks:

- The NITOS API
- The NITOS driver inside the Generic SFA Wrapper
- The MySlice plugin

NITOS API is an XMLRPC API that exposes all the information of NITOS Scheduler database. From a higher - level view, an experimenter uses MySlice - an SFA client - to discover all the resources of the federation and reserve some of them. Then through the Generic SFA Wrapper, NITOS API is called in order to get the resources. The last step is just some sql queries from the NITOS API to the NITOS Scheduler database.

The NITOS Scheduler, the SFA architecture and the MySlice are the frameworks we took advantage of and will be further described at sections 2,3 and 4 accordingly. At section 5 of this Final Project Dissertation, lies the procedure we followed in order to make NITOS testbed SFA – compliant, in more details.

1 Introduction

The inherent inability of simulation models to adequately express factors such as wireless signal propagation etc., can lead to incomplete evaluation of new protocols and applications. Thus, testing of proposed schemes under real-life settings has become the de - facto validation process.

In this direction, experimentation platforms, commonly known as testbeds, are being widely developed in order to satisfy the insistent demand from research community for services that will not only allow for efficient use of testbeds' resources, but also will provide means for the experimentation definition and performance evaluation.

Experimentation Methods	*Characteristics*
Simulation	Closed Environment No interaction with external environment System/Network is abstracted at all
Emulation	Semi - Open environment Both Real and Modeled System/Network elements are used
Real Testbed	Open environment All elements are real

Table 1: Experimentation Methods Comparison

Moving from simulation to real testbed experimentation more realism rather than abstraction is experienced, while also advantages of applicability on real case scenarios are offered. Moreover, while in simulation/emulation, scalability and reproducibility of experimentation is more likely to be controlled, in a real testbed, a service/framework that can provide means for scalability and reproducibility becomes more enticing for offering advanced instrumentation features. In Figure 1, a classification on validation of experimentation ways is illustrated and a short comparison among them is given in Table 1, while also in Figure 2 the features that a framework for real wireless testbed like NITOS [2] should support, are presented.

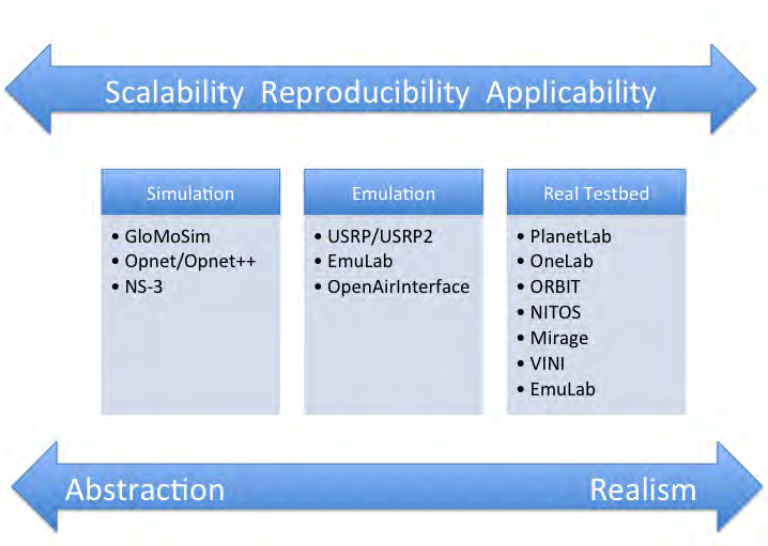


Figure 1: Cross comparison

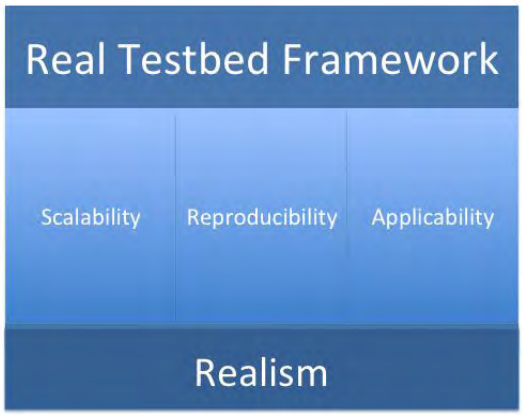


Figure 2: Classification of Validation Platforms

1.1 Need for federation of testbeds

There are many testbeds deployed and used by research teams worldwide, offering different types of resources ranging from wireless & mobile devices to virtual overlay networks. One of the challenging issues for an institution providing a testbed is to balance its setup and operational costs with the feature and technology requirements of its users. Indeed, researchers often require more experimental capabilities than what is offered by a testbed due to high hardware and operating cost. One solution to this challenge is to allow researchers to concurrently perform experiments using shared resources from testbeds provided by different institutions.

This maximizes the utilization of resources within a given testbed and extends the range of experimental capabilities for all researchers.

Use of testbeds in networking as evaluation platforms has steadily increased over the last few years as highlighted by the increased number of testbed facilities developed, offering their resources and making available the use of innovative heterogeneous technologies towards experimentation. Similar to the increased testbed availability, the increased demand from research community for designing experiments featuring heterogeneous resources, has given rise to management efforts to overcome shortcomings and confounding factors and conclude to an architecture that enables testbeds of different technologies and/or belonging to different administrative domains, to federate without losing control of their resources.

To achieve this approach, different testbed-provider institutions have to agree on and deploy some common sets of procedures, policies and tools to provide a unified testbed to the researchers, while still preserving each other's rights on the management of its contributed resources. Thus these institutions have to implement a federation of testbeds. This way, the experimenters will combine all available resources and run advanced networking experiments of significant scale and diversity. This thesis focuses on the technical aspects of making NITOS testbed part of the federation community.

2 NITOS Scheduler

The NITOS scheduler is mainly consisted of two parts: a user interface, which is responsible for guiding the user through the reservation process making sure that he does not make a reservation conflicting with reservations made by other users, and a system component, which controls the slices by ensuring that this user's experiments will use only the reserved resources. The user interface is illustrated in Figure 3.

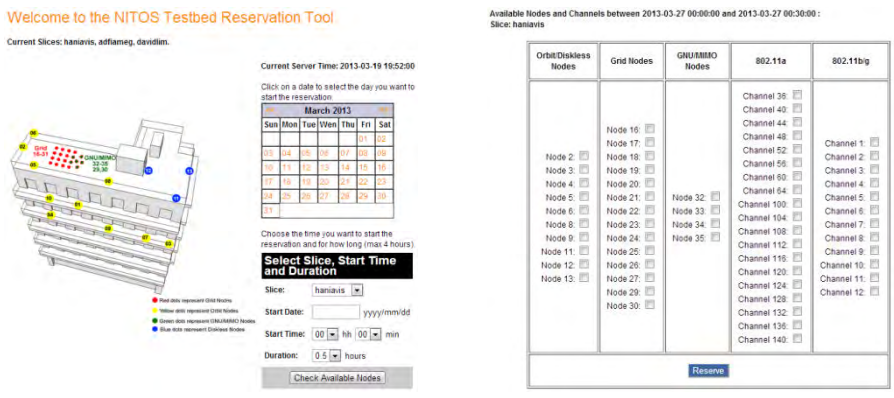


Figure 3: Resources Reservations

NITOS scheduler relies its functionality on the contrOl and Management Framework (OMF) [15] architecture. As we know, in OMF the EC communicates with the resources through an XMPP server. When an OEDL script is executed, the EC sends the experiment commands to the RC at the nodes through the XMPP server. In the default OMF configuration there is only a single slice (for running experiments) at the XMPP server called “default slice". This slice contains all testbed nodes, so every user can run experiments with every node in the testbed, provided he/she is logged in the testbed's server. What we have done is that the NITOS scheduler creates a slice at the XMPP server for each user of NITOS and we add and remove resources to/from these slices dynamically, each time a reservation starts or ends. In
NITOS testbed there is no "default slice" anymore. When we create an account for a user at the NITOS server, we also create a slice at the XMPP server with the same name (with the help of omf_create_psnode command), i.e. OMF/username. When a user reserves a node for his/her slice from the web reservation tool and for a specific time slot, the reservation entry is stored in scheduler's database. At minutes 28 and 58 (reservation time-slots at NITOS have a 30min granularity) a scheduler script is executed (via crontab) and checks the database for expiring and starting reservations. If an expiring or starting reservation is found, then the scheduler removes/adds the reservation's node(s) from/to the respective user's slice at the XMPP server (through the omf create psnode command again). With this setup only a user that has reserved a node, can run experiments with it. To do so, he/she runs "omf exec" at the server using the {slice option. But still, this setup alone does not solve other authorization problems, related to ssh-access on the nodes, rebooting of nodes and burning images on non-reserved nodes using the standard OMF procedure (omf load/save).
At this point, we denote that we have developed a security mechanism for NITOS, which is based on security mechanisms provided by the operating system (network _ltering with iptables) taking input from the scheduler database. In detail, this security mechanism works as follows: In general, the user accounts are disabled on the server and the default access policy for all users to the testbed nodes is deny. When a user makes a reservation, the system scheduler schedules a change on the firewall rules allowing only the user who made the reservation to have access to his nodes. The policy of allowing access only to the reserved nodes prevents experimenters from logging in testbed nodes that do not belong to them (either accidentally or on purpose) and affect the experiments execution. Regarding the omf procedures of load/save and tell, we have also modified OMF in order to check the scheduler's database before executing each command. This way only a user who has reserved his nodes can power on them, with "omf tell”, load an image on them, with "omf load” and save an image from them, with "omf save”.

Apart from isolation of user access to the nodes, frequency slicing is also fundamental to achieving experiment isolation in a wireless environment. Regarding frequency slicing, we have developed a solution that works even in the case where a user does not use OMF to run his experiments. Essentially, it constitutes of checking the channels being used at each resource periodically. A script runs periodically at the server (through crontab), which checks the nodes for the frequency they use

(through iwlist). In order for this to work, the server's public rsa key must be in the authorized keys file in each node's image (so that ssh - access is granted to the password-protected images of the nodes). Therefore users are obligated to copy this key in their image's file system or just not delete it, if they are using a modified version of the default NITOS baseline image. Misbehavior incidents (use of unreserved frequencies or use of images without the required rsa key) are logged in a file that a parser is checking. Then the desired policy can be applied for misbehaving users (user warning through mail, frequency switching, node rebooting etc.) A set of sniffer nodes, collectively spanning the testbed range, will also be added in the near future, in order to check for misbehavior. They will be available also for a scheduled sniffing to the experimenters, apart the administrators of the testbed.

Summarizing, from a NITOS user's perspective the steps that he/she has to follow the first time are:

- reserve nodes for his/her slice using the web application
- login to the NITOS server
- power on the reserved nodes with the "omf tell" command
- burn baseline image with the omf load command
- ssh to the node
- change slice name in the RC configuration file to match the slice
- save the image through the "omf save" command and, optionally, rename the saved image to a meaningful name.

After these steps he/she can load and save his/her image with the omf commands or run an experiment with "omf exec" command.

3 SFA

The Slice-based Federation Architecture (SFA), is a promising effort to enable attempts for testbeds federation, grew as a GENI [3] initiative. In [4], the authors defined SFA based on their experience with Emulab [6], PlanetLab [7] and VINI [8].

SFA has been designed to provide a minimal set of functionalities, a thin waist if you will that a testbed can implement in order to enter into a global and interoperable federation. An experimenter in an SFA-based environment can transparently browse resources on any federated testbed, and allocate and reserve those resources. Because of the potential for a very large number of testbeds, a global federation architecture faces a serious scalability issue. SFA introduces a fully distributed solution in which each peer testbed serves as the authority of reference for the resources that it brings, and each user community, along with its experiments, is represented by an authority (possibly, but not necessarily identified with an individual testbed).

As the name suggests, SFA is built around the central notion of a slice, which is the basic element for mapping experimenters to resources; it defines a naming scheme

that implements secure authentication within a PKI-like trust chain using X509 certificates; leveraging all this, it specifies an XMLRPC-over-https API that addresses the control plane of the federated testbed; in this respect, SFA lacks a dynamic view, and offers more help for provisioning than for running experiments. SFA describes different entities in terms of roles for experimenters, operators and owners, that are managed in an essentially decentralized architecture.

Under the SFA architecture, there is a separation between what is generic and what testbed-specific is. Testbed-specific information is captured in a resource model, called a resource specification (RSpec), which is an XML transported by the SFA layer. SFA itself does not cover such aspects as resource model, policies, reservations or measurements. These functionalities should instead be implemented on top of SFA.

History

SFA was conceived of by Larry Peterson of Princeton University in the context of efforts to create a global PlanetLab federation. The first federation of computer networking testbeds was set up between Princeton's PlanetLab Central and UPMC's and INRIA's PlanetLab Europe, starting in 2006, as part of the EU's OneLab project. This initial federation was based on the pragmatic solution of synchronizing the central databases of each of the federated entities. In this way, users of each testbed gained full access to the resources of the other. As this solution worked in the context of two peers, but was clearly not scalable, SFA pointed the way forward. The first working deployment of SFA code was developed jointly by Princeton and INRIA, the latter working in the context of the EU's OneLab2 project. Starting in 2008, SFA was used to extend PlanetLab federation to other peers, such as PlanetLab Japan and EmanicsLab. Simultaneously, SFA was adopted as a control plane architecture by GENI, in which context written specifications were drafted. Our description of SFA draws upon both the working code (from PlanetLab and from other, more recent, SFA implementations) and the written specification. These differ somewhat in their details, but agree on most of the main aspects.

In [9, 10], the authors described a solution to build a federation between a heterogeneous set of testbeds as well as means to describe their resources. In particular, they focused on the federation between SFA-like testbeds and Panlab-like testbeds. This solution proposes the intensive use of the TEAGLE framework and portal in order to control and set up the federated resources [9]. This orchestration is made possible on the Panlab side by the implementation of an SFA adapter acting as a bridge between the SFA and the Panlab internal communication [9], while in [10], the authors extended their previous work with general resource descriptions. This proposal allows to effectively run experiments over a federated Panlab/PlanetLab testbed but it is still missing some fundamental bricks for a more generic and open federation of testbeds such as the authentication and authorisation as required in [4]. Furthermore, it has a single point of failure as the TEAGLE portal remains the sole entry to the system.

A document of SFA v2.0 can be found at GENI's portal and an implementation of the SFA is currently available from PlanetLab Europe under the name of Generic SFA Wrapper [5].

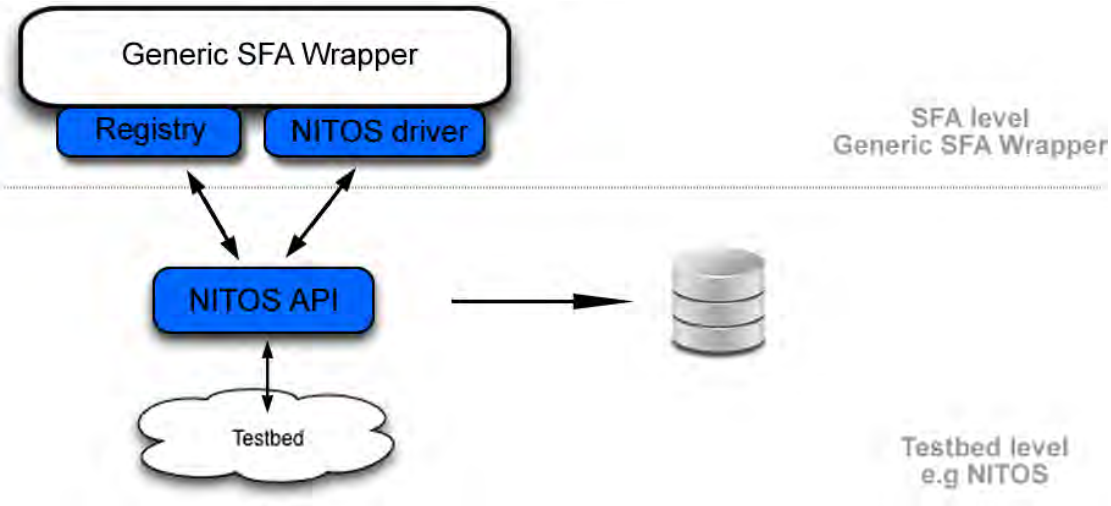


Figure 4: SFA Architecture

In order for a testbed owner to federate its testbed under the Generic SFA Wrapper, all he has to do is to implement his testbed specific driver which connects the wrapper with the testbed as shown in Figure 4. After that, any SFA client can contact the testbed and ask for information about its resources through the RSpecs scheme which is well defined in GENI and the latest version, which is under development right now, is the v3.0.

4 MySlice

MySlice is an ambitious project aiming to support researchers throughout the lifecycle of experiments that can run on a variety of testbeds spanning different administrative domains and networking technologies. Its basic principle is to bring together available **resources** with **useful information** (characteristics, performance, network measurements).

MySlice initiative started in Janury 2011 by offering annotation services for the first federated experimental resources. Today, MySlice has taken a big step toward becoming a stand-alone web framework, which will present all available resources from testbeds across the world, interconnected through the Slice-based Facility Architecture (SFA) and annotated by the TopHat measurement system.

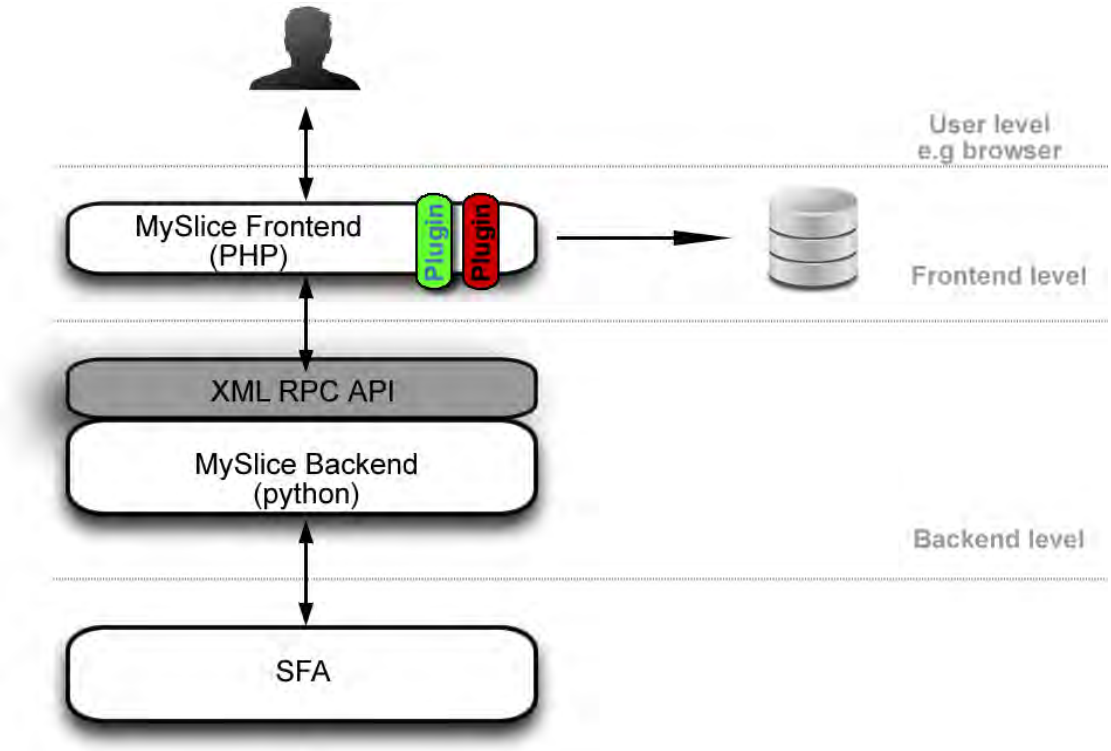


Figure 5: MySlice Architecture

MySlice framework is built with standard programming tools (php and javascript for the front-end and python for the back-end) and has a modular structure based on the concept of plugins for implementing different core functionalities (query editing, data display, and resource allocation) as illustrated in Figure 5.

MySlice is a resource management tool that makes it easy to list, filter, and reserve resources made available through the SFA control framework, annotated with useful information from different monitoring sources (reliability and utilization over time, geographic and network location etc).

Furthermore, MySlice:

- **Enables you to offer your testbed resources** to thousands of researchers worldwide
- **Allows you to share, adopt and scale up plugins and tools** that you or someone else has developed to a particular testbed.
- **Lowers the barriers** for your testbed to enter into and benefit from the global SFA federation

MySlice web framework provides a modular implementation of independent plugins and a message passing interface shared between them. They are divided in three main categories:

1) **query editing**, the type of resources that the user is interested in.

2) **data display**, the visualization of resources that match the selected query.

3) **resource allocation**, the selection and reservation (if needed) of resources.

The goal is to enable developers with expertise on different testbed technologies and different experimental practices to work in parallel for optimizing the tools presented to the users allowing them for a wide range of choices according to their own requirements. Opening this way the development of web-based user tools for experimentation and sharing effort and information can increase significantly the chances for the achievement of our challenging objective, which is a safe portal for provisioning heterogeneous resources from different federated testbeds.

5 Making NITOS SFA-Compliant

In order to bring NITOS testbed into the federation community through SFA, we decided to use the Generic SFA Wrapper [5], which is an implementation of SFA. The steps that had to be done were as follows:

- Design and implementation of NITOS API
- Design and implementation of NITOS driver for the Generic SFA Wrapper
- Design and implementation of NITOS Scheduler plugin for MySlice

During the following sections we will further analyze each step and explain the final architecture that is depicted in Figure 6 (the NITOS - specific parts of the architecture are illustrated with blue color).

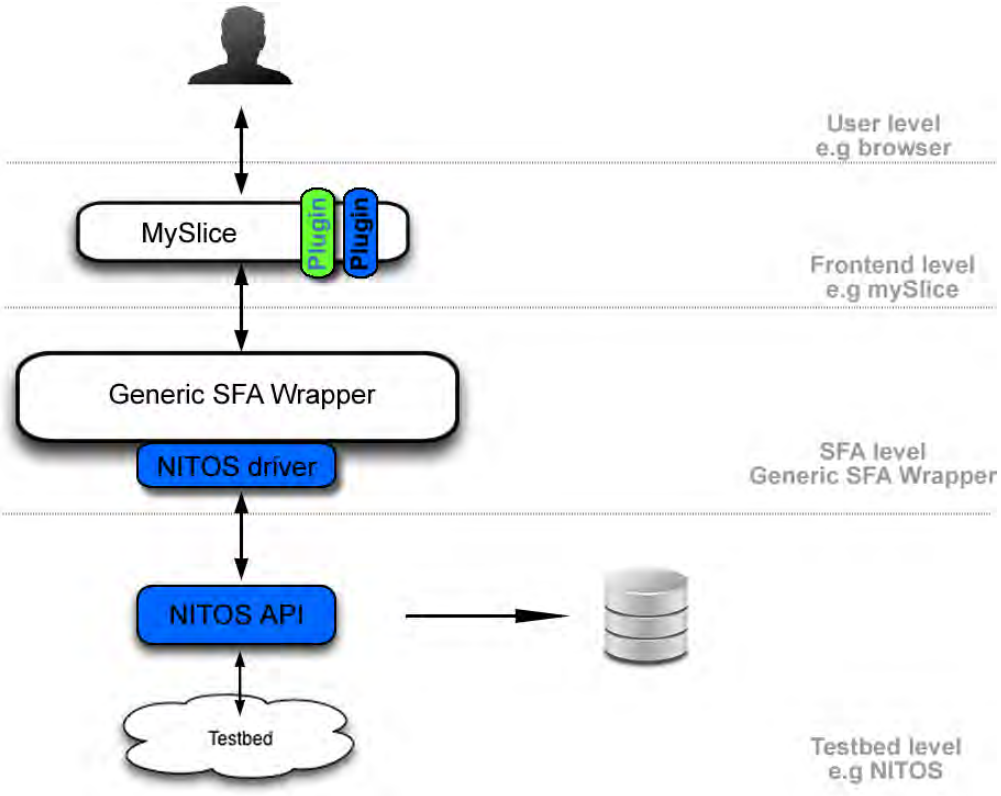


Figure 6: Making NITOS SFA - compliant

5.1 NITOS API

In the testbeds' world, there is the need of provisioning your resources and other information you would like to share with the community, in a proper way. Towards this direction, we made an API [1] that exposes all the information of NITOS Scheduler database. This API is a web service, following the rules of the XML RPC protocol in order to be client independent. It is written in the ruby language and waits for requests at the port 8081 of NITOS server. This way, any XML RPC client is able to make calls to the implemented methods of our API and get responses, if it is allowed to.

5.1.1 Information to expose

The stored information in the NITOS Scheduler database is:

- The users (apparently these are the joomla users of NITOS web interface)
- The slices
- The resources – both nodes and channels
- The reservation information for the nodes and for the channels
- Table responsible for association between users and slices

A user who is willing to have access to this information, will use an SFA client to call the Generic SFA Wrapper and after that, the Generic SFA Wrapper will call NITOS API

to discover the resources and all the information about them, reserve them, update them or delete them. The last step of the procedure is some sql queries from the NITOS API to the NITOS Scheduler database.

5.1.2 Methods Explanation

The design process of the methods that would be integrated in the NITOS API was the most crucial, as we intended to build an API as abstract as it could be, in order to be used from different OMF testbeds.

Below is shown the explanation of each method in 4 tables, one for each category.

GET methods	Explanation
getNodes	Returns the nodes
getChannels	Returns the channels
getTestbedInfo	Returns useful information about the testbed
getReservedNodes	Returns the reserved nodes for a specific time slot
getReservedChannels	Returns the reserved channels for a specific time slot
getSlices	Returns the slices
getUsers	Returns the Users

ADD methods	Explanation
reserveNodes	Reserves an array of nodes and returns the array of nodes that were successfully reserved
reserveChannels	Reserves an array of channels and returns the array of channels that were successfully reserved
addUser	Adds a user to the database and returns the id
addUserToSlice	Associates a user with a slice
addUserKey	Associates a public rsa key with a user
addSlice	Adds a slice to the database and returns the id
addNode	Adds a node to the database and returns the id
addChannel	Adds a channel to the database and returns the id

DELETE methods	Explanation
deleteKey	Deletes a public rsa key
deleteNode	Deletes a node
deleteUser	Deletes a public rsa key
deleteUserFromSlice	Disassociates a user from a slice
deleteSlice	Deletes a slice
deleteChannel	Deletes a channel

releaseNodes	Deletes an array of entries in the nodes reservation table and returns the ids that were successfully deleted
releaseChannels	Deletes an array of entries in the channels reservation table and returns the ids that were successfully deleted

UPDATE methods	Explanation
updateNode	Updates the node's information in the database
updateChannel	Updates the channel's information in the database
updateUser	Updates the user's information in the database
updateSlice	Updates the channel's information in the database
updateReservedNodes	Updates an array of entries in the nodes reservation table and returns the ids that were successfully updated
updateReservedChannels	Updates an array of entries in the channels reservation table and returns the ids that were successfully updated

5.1.3 Methods Definition

We have implemented 4 categories of methods that can be called from an XML RPC client. **GET** methods, **ADD** methods, **DELETE** methods and **UPDATE** methods. When a client makes a method call, he should follow this generic structure "Auth,filter,retValue" for the GET methods and this "Auth,filter", for the other categories. **Auth** is for authentication purposes, **filter** is for filtering the request and is a *struct*, **retValue** is for specifying which particular value to return and is an *array*.

In the following tables, is presented the definition of each method and the Returned Value.

GET methods	Default Returned Value
getNodes(Auth, filter, retValue)	[{hostname,node_id,node_type,floor,view,wall,position:{X,Y,Z}},…]
getChannels(Auth, filter, retValue)	[{channel_id, modulation, channel, frequency}, …]
getTestbedInfo(Auth, filter, retValue)	[{name,grain,OMF_version,scheduler_version,gw_address,longtitude,latitude}]
getReservedNodes(Auth, filter, retValue)	[{reservation_id, slice_id, start_time, end_time, node_id},…]
getReservedChannels(Auth, filter, retValue)	[{reservation_id, slice_id, start_time, end_time, channel_id}, …]

getSlices(Auth, filter, retValue	[{slice_id, slice_name, user_id=[[“”],[“”]]},…]
getUsers(Auth, filter, retValue)	[{user_id, username, email, keys=[[“”],[“”]]},…]

ADD methods	Default Returned Value
reserveNodes(Auth, {slice_id,start_time,end_time,nodes:[node_id,…]})	[node_id, …]
reserveChannels(Auth, {slice_id,start_time,end_time,channels:[channel_id,…]})	[channel_id, …]
addUser(Auth, {username,email})	user_id
addUserToSlice(Auth, {slice_id,user_id})	0 OR -1
addUserKey(Auth, {user_id,key})	0 OR -1
addSlice(Auth, {slice_name})	slice_id
addNode(Auth, {name,node_type,floor,view,wall,position:{‘X’,’Y’,’Z’}})	node_id
addChannel(Auth, {channel,frequency,modulation})	channel_id

DELETE methods	Default Returned Value
deleteKey(Auth, {key})	0 OR -1
deleteNode (Auth, {node_id})	0 OR -1
deleteUser (Auth, {user_id})	0 OR -1
deleteUserFromSlice (Auth, {user_id,slice_id})	0 OR -1
deleteSlice (Auth, {slice_id})	0 OR -1
deleteChannel (Auth, {channel_id})	0 OR -1
releaseNodes (Auth, {reservation_ids:[reservation_id,…]})	[reservation_id, …]
releaseChannels(Auth,{reservation_ids:[reservation_id,…]})	[reservation_id, …]

UPDATE methods	Default Returned Value
updateNode(Auth, {node_id,fields:{field_name:value}})	0 OR -1
updateChannel(Auth, {channel_id,fields:{field_name:value}})	0 OR -1
updateUser(Auth, {user_id,fields:{field_name:value}})	0 OR -1
updateSlice(Auth, {slice_id,fields:{field_name:value}})	0 OR -1
updateReservedNodes(Auth, {reservation_ids:[reservation_id,…],start_time:value,end_time:value})	[reservation_id, …]
updateReservedChannels(Auth, {reservation_ids:[reservation_id,…],start_time:value,end_time:value})	[reservation_id, …]

5.2 Generic SFA Wrapper

The architecture we adopted towards the goal of federating NITOS testbed is the SFA [4] and in particular its implementation the Generic SFA Wrapper [5].

To achieve the objective of federating different heterogeneous testbeds, SFA defines a distributed and secure API that allows researchers affiliated with an administrative domain belonging to a federation to browse all the available resources and allocate those required to perform a specific experiment. Note that SFA is more of a specification of a standard, rather than a specific implementation and actually there are different versions of it today, tailored to address the need of different technologies (PlanetLab, ProtoGENI, OpenFlow).

The development team of INRIA research centre in France, created the Generic SFA Wrapper that stems from few reference implementations (PLC, PLE, ProtoGENI etc.). The proposed SFA Wrapper is able to be implemented in two independent modules as illustrated in Figure 7: 1) a generic SFA implementation which is the same for all technologies and 2) a testbed-specific driver, which is responsible for implementing the local resource browsing and allocation functionality.

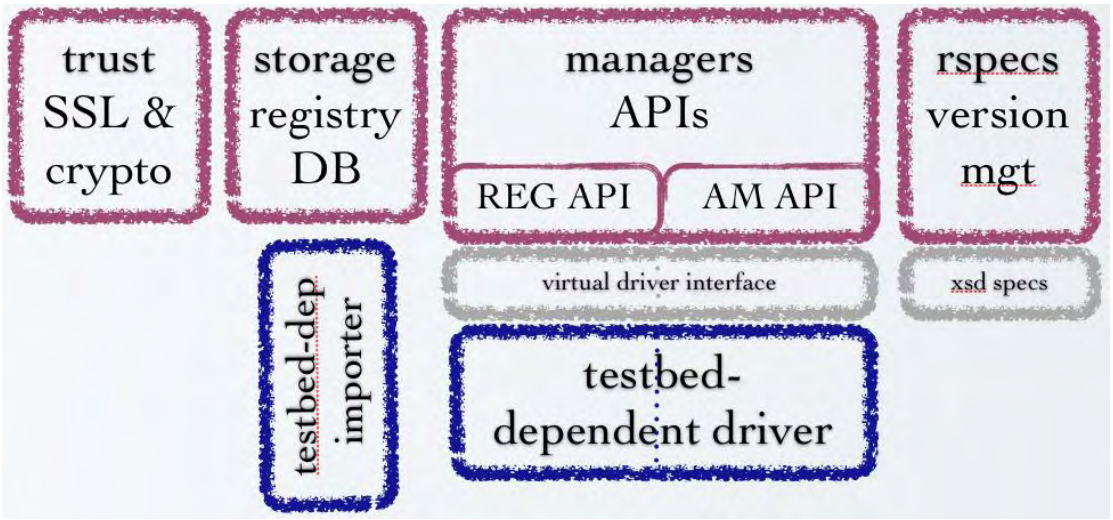


Figure 7: Generic SFA Wrapper architecture

This work in progress will soon allow a variety of heterogeneous testbeds which are cooperating in the context of several European projects, to plug their resources into the global SFA federation and enable them to both attract external users and allow their internal users to access a big list of heterogeneous resources.

The Generic SFA Wrapper is an open – source software that allows you to federate your testbed into the emerging SFA – based global federation of testbeds. More specifically it is a secure, distributed and scalable narrow waist of functionality that enables testbeds of different technologies and/or belonging to different administrative domains to federate without losing control of their resources. This will allow researchers to combine all available resources and run advanced networking experiments of significant scale and diversity.

It builds on the Slice-based Federation Architecture, that is in the process of being adopted in the GENI and FIRE research communities in the US and Europe- as well as elsewhere in the world and several dozens of testbeds either use it or are in the process of adopting it.

5.2.1 Benefits

Being a testbed owner and adopting the Generic SFA Wrapper, you enjoy several privileges like:

- It lowers the barriers for your testbed to enter into the global SFA federation
- SFA-enables your testbed by providing the web services interface, as well as handles the coding and parsing and many other details
- It enables you to join dozens of testbeds worldwide that are adopting SFA and benefit from mutualised drivers, documentation and support services

SFA provides a secure, distributed and scalable narrow waist of functionality for federating heterogeneous testbeds. However, there are barriers to the entrance of using SFA: a testbed owner would normally need to implement the certificate-based authentication and authorization mechanisms used by SFA; also, they would need to write coders and parsers for files that describe the resources on their testbed; this can be laborious.

By adopting SFA through the SFA Wrapper you can, not only offer your testbed through a global, testbed federation, but also benefit from the tools and controls that the SFA federation offers.

5.2.2 Deploying

If a testbed owner wants to “wrap” his testbed, he needs to do at most three things:

- If his testbed does not already have a web services interface, you need to provide one, through which SFAWrap can interact with his testbed.
- If his testbed has a web services interface that is different from those of other testbeds already into the federation, then he needs to write a driver for SFAWrap. This translates SFA originated queries into queries for his testbed. There are a lot of existing drivers that can act as a protoype for developing a new from scratch.
- If his testbed has special resources that are not captured by an existing resource description file format, then he needs to extend SFAWrap’s parser to encode and decode these resource specifications, known as RSpecs. Again, he can base his work on existing examples.

A Linux installation package for the Generic SFA Wrapper can be found here.

5.2.3 RSpecs

The SFA gives users access to heterogeneous resource types. The RSpecs is the means that the SFA uses for declaring which resources a user wants on each Aggregate.

The ProtoGENI RSpec is our mechanism for advertising, requesting, and describing the resources used by experimenters. We derived many of the basic principles from our previous format used in assign, the network mapper used in Emulab. The format has the advantage of having a fairly simple structure and allows us to draw.

The RSpec has three distinct purposes and therefore RSpecs are divided up into three different closely-related languages to address each of these purposes in particular.

- Advertisements are used to describe the resources available on a Component Manager. They contain information used by clients to choose resources (components). Other kinds of information (MAC addresses, hostnames, etc.) which are not used to select resources should not be in the Advertisement.
- Requests specify which resources a client is selecting from Component Managers. They contain a (perhaps incomplete) mapping between physical components and abstract nodes and links.
- Manifests provide useful information about the slices actually allocated by a Component Manager to a client. This involves information that may not be

known until the slice is actually created (i.e. dynamically assigned IP addresses, hostnames), or additional configuration options provided to a client.

In order to expose properly our resources and all the reservation information, we defined NITOS RSpecs in cooperation with INRIA team, which are v1-like but including reservation information. Below there is an example of a NITOS advertisement RSpec, where we can see all the nodes, following the channels and at the end the active leases. With active leases, we refer to the reservations that exist from now on.

Example of a NITOS advertisement RSpecs:

```
<?xml version="1.0"?>
<RSpec type="NITOS" expires="2013-01-30T10:26:27Z" generated="2013-
01-30T09:26:27Z">
  <network name="omf">
    <node component_manager_id="urn:publicid:IDN+omf+authority+cm"
component_id="urn:publicid:IDN+omf:nitos+node+node001"
component_name="node001"
site_id="urn:publicid:IDN+omf:nitos+authority+sa">
      <hostname>node001</hostname>
      <location country="unknown" longitude="39.360839"
latitude="22.949989"/>
      <position_3d x="1" y="1" z="5"/>
      <exclusive>TRUE</exclusive>
      <gateway>nitlab.inf.uth.gr</gateway>
      <granularity>1800</granularity>
      <hardware_type>orbit</hardware_type>
    </node>
    <node component_manager_id="urn:publicid:IDN+omf+authority+cm"
component_id="urn:publicid:IDN+omf:nitos+node+node002"
component_name="node002"
site_id="urn:publicid:IDN+omf:nitos+authority+sa">
      <hostname>node002</hostname>
      <location country="unknown" longitude="39.360839"
latitude="22.949989"/>
      <position_3d x="1" y="2" z="6"/>
      <exclusive>TRUE</exclusive>
      <gateway>nitlab.inf.uth.gr</gateway>
      <granularity>1800</granularity>
      <hardware_type>orbit</hardware_type>
    </node>
    <node component_manager_id="urn:publicid:IDN+omf+authority+cm"
component_id="urn:publicid:IDN+omf:nitos+node+node003"
component_name="node003"
site_id="urn:publicid:IDN+omf:nitos+authority+sa">
      <hostname>node003</hostname>
```

```
      <location country="unknown" longitude="39.360839"
latitude="22.949989"/>
      <position_3d x="1" y="3" z="4"/>
      <exclusive>TRUE</exclusive>
      <gateway>nitlab.inf.uth.gr</gateway>
      <granularity>1800</granularity>
      <hardware_type>orbit</hardware_type>
    </node>
.
.
.
<spectrum>
      <channel channel_num="36" frequency="5180.0"
standard="IEEE802_11a"/>
      <channel channel_num="40" frequency="5200.0"
standard="IEEE802_11a"/>
      <channel channel_num="44" frequency="5220.0"
standard="IEEE802_11a"/>
      <channel channel_num="48" frequency="5240.0"
standard="IEEE802_11a"/>
      <channel channel_num="52" frequency="5260.0"
standard="IEEE802_11a"/>
      <channel channel_num="56" frequency="5280.0"
standard="IEEE802_11a"/>
.
.
.
<lease slice_id="urn:publicid:IDN+omf:nitos+slice+plenitos"
start_time="1363032000" duration="8">
      <node component_id="urn:publicid:IDN+omf:nitos+node+node002"/>
      <node component_id="urn:publicid:IDN+omf:nitos+node+node001"/>
      <channel channel_num="1"/>
      <channel channel_num="2"/>
    </lease>
    <lease slice_id="urn:publicid:IDN+omf:nitos+slice+tribino"
start_time="1359529200" duration="8">
      <node component_id="urn:publicid:IDN+omf:nitos+node+node019"/>
      <node component_id="urn:publicid:IDN+omf:nitos+node+node020"/>
      <node component_id="urn:publicid:IDN+omf:nitos+node+node021"/>
      <channel channel_num="5"/>
      <channel channel_num="6"/>
      <channel channel_num="7"/>
    </lease>
</network>
</RSpec>
```

5.3 Developing a plugin for MySlice

5.3.1 Installing and configuring the frontend

The installation of MySlice frontend is quite easy as there is the debian package *myslice-joomla,* which installs all required code in order to be able to call the MySlice backend. The plugin developer can take advantage of the already existing plugins and start from scratch to implement his own.

You can choose whether or not to install the MySlice backend on your server, by configuring the url and the port that the frontend will point to. In our implementation, we use the backend instance that is deployed on *demo.myslice.info/7080.*

Overview

In general, the different platforms interconnected will all have different authentication schemes, and thus will require different tokens.

Myslice is designed to support multiple users. It can be used either locally (python library) or remotely (xmlrpc api, web GUI). In a local setup, a user does not need to authenticate to Myslice, while this is made mandatory for remote access by the API.

The following figure succinctly presents how platforms, users and accounts are handled by Myslice.

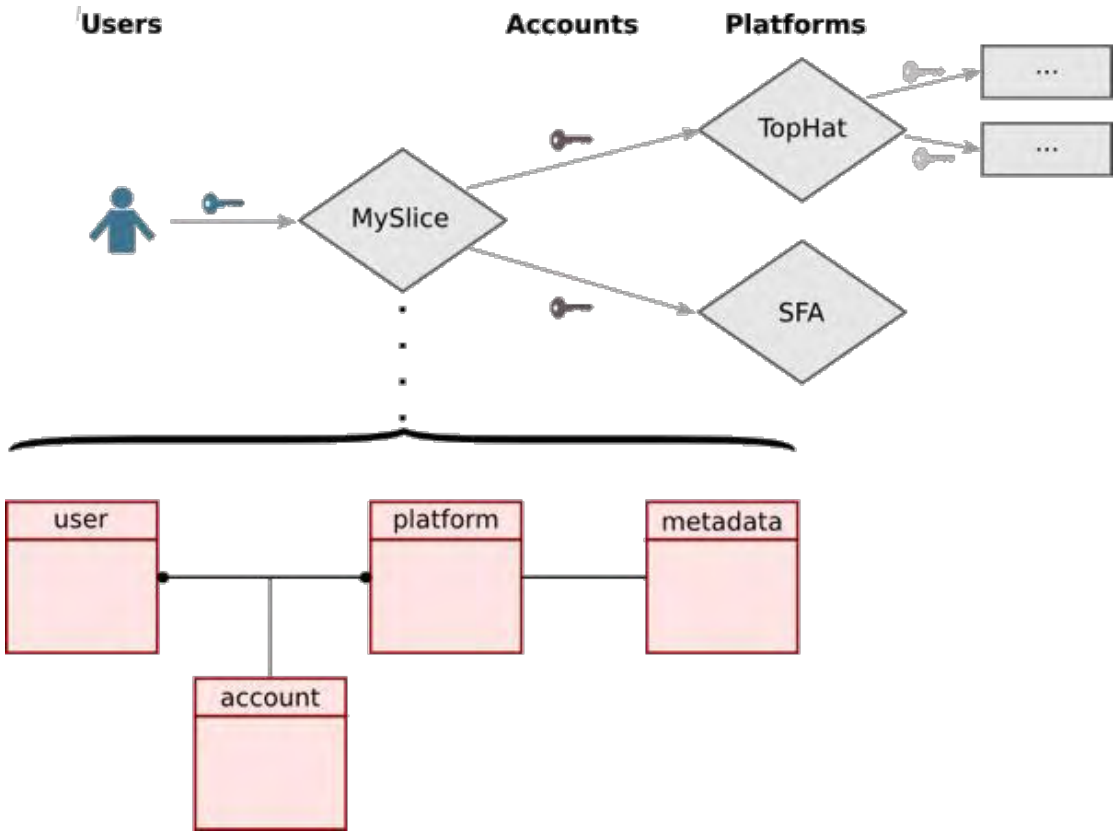


Figure 8: MySlice overview

This functionality is used when users are remotely accessing Myslice, either through the web GUI or the API.

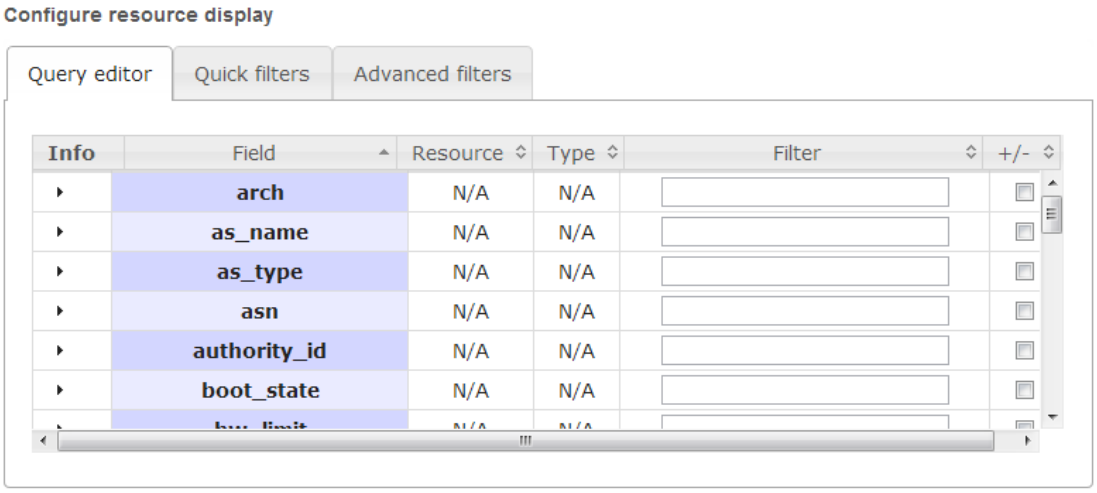
Currently three authentication methods are supported:

- users have a local account
- users enter their OneLab/PlanetLab Europe or PlanetLab Central authentication token:
- users enter their signed SFA certificate in their browser (only supported by the web GUI)

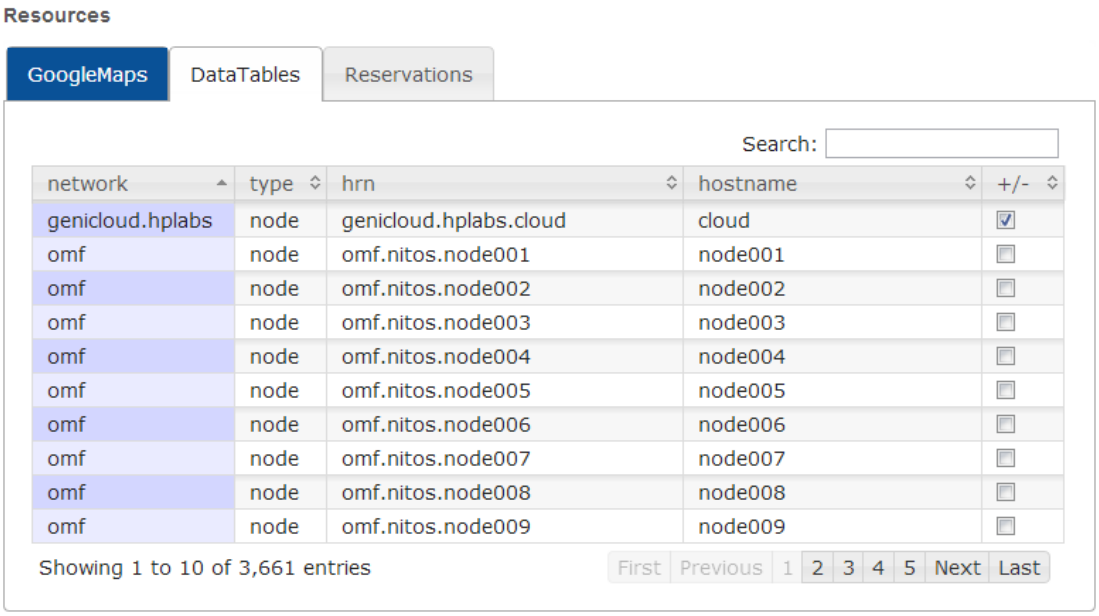
This scheme could be extended to more advanced solutions in the future, should the need arise.

5.3.2 Frontend structure

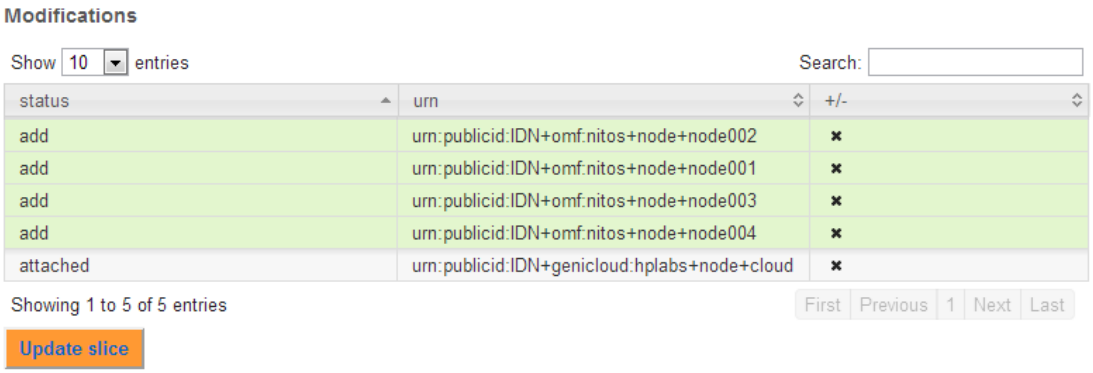
The architecture of the frontend of MySlice is separated in three sections as depicted in Figure 9. The **Filter** plugins are those that are shown on top, following the **Display** plugins and at the bottom lie the **Resource Allocation** plugins. All these are communicating between each other using a publish – subscribe mechanism.



(a) Filter



(b) Display



(c) Resource Allocation

Figure 9: MySlice frontend

5.3.3 NITOS Scheduler plugin

Regarding the NITOS Scheduler plugin we implemented, it belongs to the "Display" category and it enables us to represent all the information that exists in the NITOS Scheduler database. As to the user interface, we relied on the previous web interface of the NITOS Scheduler [14] as depicted in Figure 10. The user is able to see the location of the nodes on our building represented in a 3D figure, in favor of choosing the best topology for him. Subsequently, he will choose some of the available time slots for each node he intends to include in his experiment.

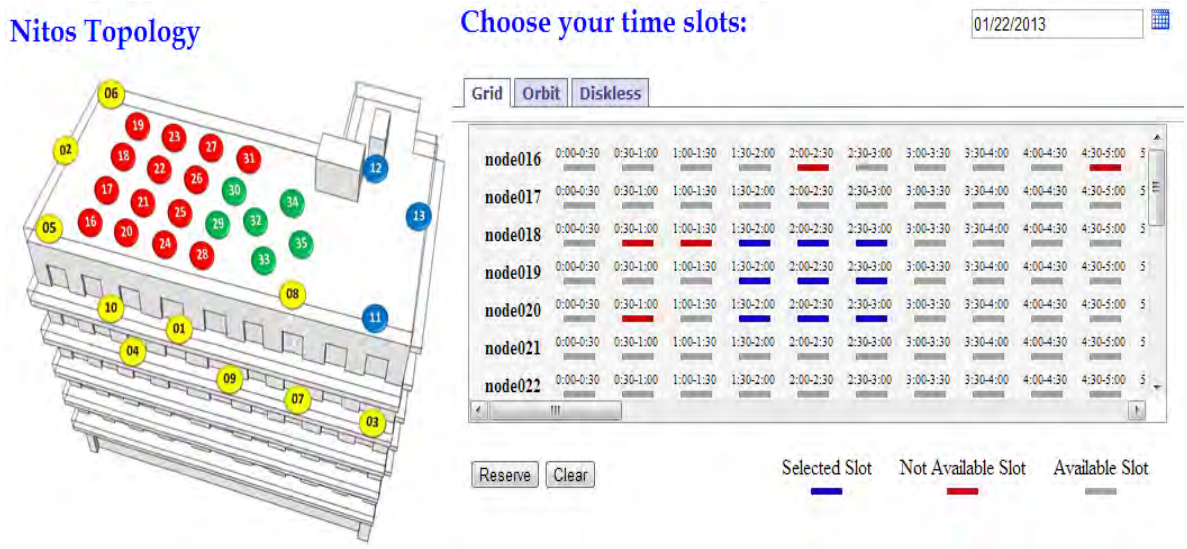


Figure 10: NITOS Scheduler plugin

Like every MySlice plugin, the NITOS Scheduler plugin consists of a php file which is responsible for the appearance together with a css file and some javascript files which are responsible for the interaction both with the backend API and the other plugins.

6 Conclusion and Future Work

In this Final Project Dissertation – Thesis we described the process of making NITOS testbed, part of the federation community.

Our basic goal is allow to experimenters from around the world to have access to NITOS resources together with heterogeneous resources from other federated testbeds. Towards this goal, we built a web service as an upper layer of NITOS testbed, in order to provision all the information of our database and to connect with the SFA. Finally, a user interface was implemented to give a nicer display of the resources.

Currently, we are in the process of improving both the NITOS API code and the NITOS Scheduler plugin, adding more functionality. This will end up to a neat NITOS Scheduler package – the API, the driver and the plugin - that will be used from several OMF testbeds which want to be part of the SFA community. In addition, we intend to take advantage of the web service we implemented, by building web applications, like an Android application.

In addition, there are more that should be done in the authentication and authorization part of the whole procedure, as currently we are not mature enough in this sector.

References

1. NITOS API, https://github.com/NitLab/Nitos_api
2. NITOS testbed, http://nitlab.inf.uth.gr/NITlab/index.php/testbed
3. GENI, Exploring Networks of the Future, http://www.geni.net/
4. L. Peterson, R. Ricci, A. Falk, and J. Chase, *"Slice-Based Federation Architecture",* Technical report, Geni, 2010
5. Generic SFA Wrapper, http://sfawrap.info/
6. B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler,C. Barb, and A. Joglekar, "*An integrated experimental environment for distributed systems and networks",* SIGOPS Oper. Syst. Rev., 36(SI):255{270, 2002
7. L. Peterson and T. Roscoe, "*The Design Principles of PlanetLab. Operating Systems Review"*, 40(1):11{16, January 2006
8. A. Bavier, N. Feamster, M. Huang, L. Peterson, and J. Rexford, *"In VINI Veritas: Realistic and Controlled Network Experimentation"*, in ACM, editor, Proc. of ACM SIGCOMM'06, September 2006
9. S. Wahle, T. Magedanz, and K. Campowsky, "*Interoperability in Heterogeneous Resource Federations"*, in Proc. of 6th International ICST Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities (TridentCom), Springer-Verlag, 2010
10. S. Wahle, C. Tranoris, S. Fox, and T. Magedanz, "*Resource Description in Large Scale Heterogeneous Resource Federations"*, in Proc. of 7th International ICST Conference on Testbeds and Research Infrastructures for the Development of Networks and communities (TridentCom), Springer-Verlag, 2011
11. OpenSFA, http://opensfa.info/
12. T. Rakotoarivelo, G. Jourjon, T. Parmantelat and T. Korakis, "*Distribute and Federate: a Framework for Networking Testbed Federation"*
13. MySlice, http://myslice.info/home
14. NITOS Scheduler web interface, http://nitlab.inf.uth.gr/NITlab/index.php/scheduler/reservation
15. OMF, http://omf.mytestbed.net/