Design, Architecture and Implementation of a Resource Discovery, Reservation and Provisioning Framework for Testbeds

Donatos Stavropoulos*†, Aris Dadoukis*†, Thierry Rakotoarivelo‡,
Max Ott‡, Thanasis Korakis*† and Leandros Tassiulas§
*Department of Electrical and Computer Engineering University of Thessaly, Greece
†Centre for Research and Technology Hellas, CERTH, Greece
‡National ICT Australia (NICTA), Alexandria, Australia
§Department of Electrical Engineering, Yale University, New Haven, USA

***Abstract*—Experimental platforms (testbeds) play a significant role in the evaluation of new and existing technologies. Their popularity has been raised lately as more and more researchers prefer experimentation over simulation as a way for acquiring more accurate results. This imposes significant challenges in testbed operators since an efficient mechanism is needed to manage the testbed's resources and provision them according to the users' needs. In this paper we describe such a framework which was implemented for the management of networking testbeds. We present the design requirements and the implementation details, along with the challenges we encountered during its operation in the NITOS testbed. Significant results were extracted through the experiences of the every day operation of the testbed's management.**

I. Introduction

Research into the Future Internet 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.

In order to make large scale experimentation feasible, reproducible and easy, several experimental tools have been developed so that to standardize the way experimenters interact with resources of the testbeds. The cOntrol and Management Framework (OMF) [1] is considered the prevalent tool for experimentation in networking testbeds, providing a modular architecture capable of controlling heterogeneous resources. The Network Experiment Programming Interface (NEPI) [2] provides another alternative to the experimenters who can leverage it, in order to orchestrate their large scale experiments. Both tools use the Federated Resource Control Protocol (FRCP) [3] in order to communicate with the underlying resources. The objective of these tools is to provide a more standard and flexible way in controlling testbed resources, in contrast to the user's custom scripts, which are usually tailor-made according to a specific testbed where the experiment is conducted.

Apart from the user tools that facilitate the experimentation workflow from the experimenter's perspective, there is also a need to have equivalent tools for management and administration of the experimental facilities. OMF in its latest release (version 6) tackled the problem of controlling the experiment/resources and in this paper we present how the management of the facility is handled by the framework. More specifically, we analyze the design principles and the overall architecture of the implemented framework, which tries to tackle the complex problem of testbed management. The ultimate objective is to aid the administrators of experimental facilities in their everyday tasks regarding facility management, as well as to provide valuable insights for those who intend to build and manage a testbed.

The rest of the paper is organized as follows: In Section II the complex problem of testbed management is analyzed and in Section III related work is given. Section IV provides a thorough view in the design principles and the architecture of the presented framework. Finally in Section V, the NITOS case study is evaluated, while a conclusion of our work and future extensions are given in Section VI.

II. Problem Statement

Managing a testbed can be a complex procedure, especially when remote access is given 24/7 to experimenters around the world, presuming that every procedure will be automated and human intervention will be limited or none at all. Running a testbed locally in a lab where the experimenter is physically present, might not need any special management software for discovering, reserving and provisioning the resources, since everything is done manually on-site by the experimenter. However, when the experimenter is not physically present, his actions on the actual resources are limited, since having another human serving this purpose is not efficient in many ways, the only viable alternative is to offer these functionalities as a service to the experimenter through software management components.

The initial step for conducting an experiment is finding the necessary resources on top of which the experiment will

be executed. During the "discovery" phase where the experimenter tries to find resources that fulfill his requirements, testbeds need to expose their list of resources with their corresponding properties to the experimenter. Some may say that having everything documented in a website can fulfill this purpose but the truth is that a more versatile solution is needed besides a static representation of the resources in a website. The list of resources should depict their actual availability and capabilities at the time the advertisement is requested. The best practice is to offer the list of resources through an inventory service to the experimenters and their tools and at the same time provide the necessary hooks so that other components could update the inventory through API calls. The benefits of having an API for exposing the resources of a testbed are multilateral, since it can be leveraged from tools like a reservation component or a monitoring tool. For instance, when a resource is reserved by an experimenter, the inventory service automatically is updated by the reservation component and informs any forthcoming experimenter about the unavailability of this resource. Furthermore, removing a resource from the list due to random failures or maintenance can be done automatically, at the time a monitoring tool of the testbed diagnoses a malfunction in one of the resources.

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 administrative tools. A critical component of every testbed's management framework is the inventory that keeps the unique characteristics of each provided resource. This component should allow extensibility and modularity in order to be able to maintain information of newly added types of resources. Additionally, a way to initialize and manage the actual resource should be adequately defined with one or more services. Therefore, the management framework should not only be capable of describing new resources, but also instantiating them as part of the provisioning phase.

Moreover, features like resource reservation and policy enforcement, as proposed in [4], are of great interest to the administrators of experimental facilities who seek ways to better utilize their platforms through fine-grained policies [5]. 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 challenges to the administrators who want to apply different policies on users originated from other testbeds of a federation.

Besides the capabilities provided for managing local testbed components, 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 Slice-based Federation Architecture (SFA) [6] protocol that is widely used in order to achieve this objective. Federations between testbeds fit the need for global scale experiments involving different types of resources. However, they add an extra burden to the administrators of the experimental facilities who want to federate with other testbeds, since they need to develop the necessary adaptation layers over or within their management software.

III. Related Work

In the context of Future Internet research facilities, a diverse number of resource types (wired and wireless networking, software defined networks, cloud infrastructure, etc) is provided to the research community, by various facilities around the world. Each one of them requires a way for managing its infrastructure and a way to seamlessly federate with other testbeds. To this end, various software components have been developed for the management purposes of testbeds.

Generic SFA wrapper (SFAWrap) [7] is a framework that creates an SFA interface, by consuming services that are built and used by a testbed for resource interaction and manipulation. In order for a testbed to utilize the SFAWrap framework, an implementation of a set of web services is required. If the testbed has already those services, it is possible to implement a driver that translates them into the supported format. In addition if the testbed has any kind of special resources, the SFAWrap's parser can be extended so that it can decode/encode those resources. SFAWrap is not a full management framework and best be used when a testbed already has a working management framework that is not SFA compatible.

FIRMA [8] is proposing an extensible architecture for testbed federation that relies on a semantic Linked Data driven information model. An initial Java framework has been implemented, mostly for evaluation purposes. An essential characteristic is their focus on standardizing the exchanged resource descriptions by leveraging semantic web technologies.

The ORCA [9] control framework is actually an umbrella project that consists of several software components that evolve together with the ORCA architecture. The ORCA framework manages programmatically controllable shared elements, which may include servers, storage, networks, or other components that constitute a Cloud computing infrastructure.

FOAM [10] is an Aggregate Manager (AM) that is part of the GENI project [11] and supports GENI v3 RSpecs. It is a complete solution that fully supports allocation and experimentation with OpenFlow resources. Likewise, the OFELIA control framework [12] which has been developed as part of the OFELIA FP7 project, is responsible for the management of OpenFlow resources. It contains AMSoil, a light-weight framework for creating Aggregate Managers. Using this framework, AMs for OpenFlow and VirtualMachine experimentation have already been built in the context of OFELIA testbeds.

Due to the fact that testbeds are heterogeneous in the context of provided resources, various solutions have been proposed for dealing with the problem of testbed management. Each of

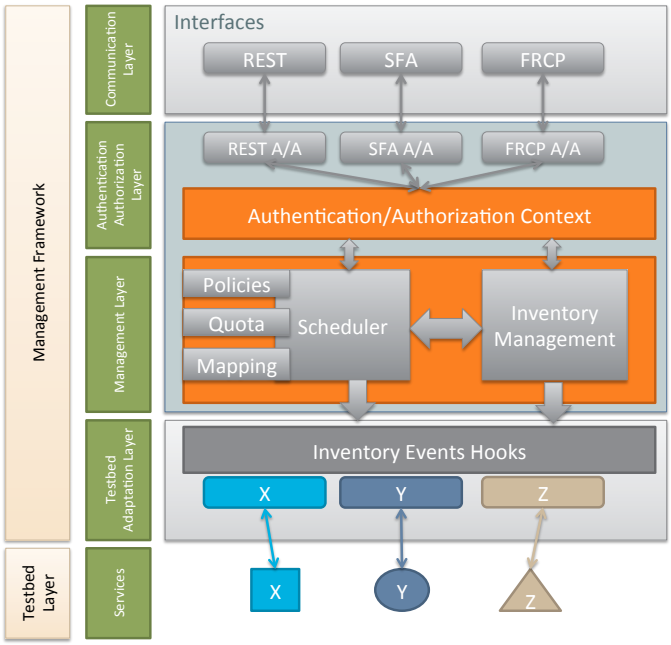

Fig. 1. Overall architecture of the implemented management framework

them has been built under the concept that a subset of similar testbeds must be first served and then strive to become as generic as possible. In our approach, we focused on avoiding the drawbacks of concentrating all the testbed functionalities in one component, the AM. Instead, we decoupled the inventory management of the testbed with the testbed's specific functionalities, exploiting this way the benefits of a Service Oriented Architecture (SOA). We believe that these decisions together with the open source nature of our implementation, will enable easier adoption of our framework from different experimental facilities. In the following section we provide more details about the design and architecture of our framework.

IV. Architecture

The design principles, along with the architecture of our framework, were formed by taking into consideration all the challenges and the desired characteristics that a management system should feature, based on the problem analyzed in Section II. We divide our framework in several fundamental architectural components that each of them plays a significant role in a specific problem area of the facility management. We enumerate these as a set of basic functionalities that an administrator would like a management framework to feature. The overall architecture with all the components can be seen in Figure 1.

A. Communication Interfaces

One essential characteristic of a management framework is its versatility in terms of communication interfaces or else APIs. In our implementation multiple communication interfaces are utilized, supporting widely used protocols in the field of testbeds like SFA and FRCP. Next to that, a REST API has also been implemented, which is one of the best current practices in the domain of web services.

The SFA endpoint is an implementation of the GENI AM API v2 [13] which is used both in GENI and FIRE testbeds as a way of federation. An upgrade to the new version (version 3) [14] is already under development and will be operating side by side with that of v2 with no complications. This will allow our framework to be reachable not only by tools that are compatible with the latest API, but also by tools that can only interact with v2 AMs.

The alternative to the SFA interface, where a formal standard API has to be followed, is the custom REST API which is flexible to be implemented according to our needs. This way we can implement any missing functionalities of the SFA through the REST interface, like the administrative management, which is not part of the SFA that primarily targets to resource management. One more difference to the SFA is that REST uses JSON serialization instead of XML used by SFA.

Finally, the FRCP interface is primarily used for interoperability with Resource Controllers (RCs) in order to automate procedures that have to do with testbed specific functionalities. It is not intended for getting a resource list or modifying the inventory. The need of FRCP is crucial since next to SFA is the other standard protocol emerged from testbed federations. SFA is used for resource discovery and provisioning, whereas FRCP for resource control. By featuring both interfaces, our framework can easily provide the "glue" between these two worlds and ease the transition from the resource provisioning phase to that of the experimentation.

B. Authentication and Authorization

Authenticating and authorizing a user is of crucial importance to any testbed management framework. In our implementation the Authentication/Authorization (A/A) module is the part where requests are accepted or denied based on an authentication and authorization context. As mentioned above, our framework features multiple communication interfaces and therefore can serve requests from a REST, an SFA, or even an FRCP interface. This implies that each interface receives a different type of credentials, which requires the utilization of different mechanisms in order to handle them successfully.

Initially, the authentication process takes place, confirming the identity of the user who has sent a request. The authentication mechanism can be configured to trust and authenticate the users of a specific set of certificate authorities. These different certificate authorities usually are different testbeds federated with the given testbed. The identification of the user is done through client side X.509 certificates in all the interfaces. However, the contents of these certificates differ in some points. For instance, the SFA X.509 certificates use an extension in order to provide a uniform resource name (URN), which enables the testbed AM to link an associated SFA request to a specific experiment that includes a set of resources. These details impose a set of different routines

per interface that are capable of handling specific attributes included in the certificates.

The phase of user Authentication, follows the initialization of the Authorization context which consists of a set of simple assertions that denote the user's permitted actions on: (i) Resources; (ii) Accounts; (iii) Reservations. These actions designate if the user is permitted to: (i) Create; (ii)View; (iii) Modify; or (iv) Release a resource, account or a reservation. In SFA requests, there is always a signed XML file containing the privileges of the user, which are mapped to the above set of assertions by the A/A module. In the REST and FRCP requests, the A/A assertions are filled based on the obtained identity of the user from the certificate and his status or role. In FRCP there are developments towards the adoption of signed assertions that could be potentially used by the corresponding routine of the A/A module for authorization context.

The goal of the A/A module is to provide the necessary means to the testbed administrator so that he could easily modify and describe his own fine grained policies. This is accomplished thanks to the discrimination of policies which are divided based on their protocol (SFA, REST, FRCP) and then mapped to a common set of rules and assertions. Hence, testbed administrators need only to modify the A/A module once for the three different APIs.

C. Scheduling

The Scheduler is a critical component since it is the part where decisions regarding resource reservation/allocation take place based on the availability and any applicable policies. More specifically, when a request for resource reservation is received and forwarded to the Scheduler component, the Scheduler decides based on the authorization context, whether to fulfill the request or to reject it.

Aside from the simple policy of whether to accept or deny a request, the Scheduler can be the point where more sophisticated policies are enforced based on quota or the roles/status of the users. Testbed administrators, only need to modify the Scheduler module in order to define their resource allocation policies. In our case, a simple First-Come-First-Served (FCFS) policy was applied to the requests for resource reservation. That does not limit the potential of implementing a different policy based on user's role/status.

Another significant functionality of the Scheduler module is the capability of mapping a set of abstract resources to the actual physical ones. In another words, users can ask for resources without specifying which ones, letting the scheduler decide for them which resources are better fulfilling their requirements. For instance, a user can ask for two wireless nodes featuring WiMAX connectivity to be reserved for 6 hours. The Scheduler will map his request to the first two available nodes that have WiMAX capabilities and send the reservation confirmation back to the user.

The aforementioned functionality has been developed in a dedicated submodule of the Scheduler in a way that testbed operators are able to import their own algorithms for mapping resource requirements to the actual physical resources. These decisions can be taken based on the utilization of the testbed, the node's characteristics or even required connectivity between the nodes. An important characteristic of a resource is whether it is virtualized or exclusive. Virtualized is the resource where multiple users can use one resource simultaneously, whereas exclusive resource is accessed by one person at a time. Our current implementation works by favoring first the less utilized testbed (in case more than one testbed is managed), then chooses exclusive resources over virtualized and the last criteria is the availability of resources in the requested timeslot.

D. Testbed Adaptation Layer

The southbound of our framework is the Testbed Adaptation Layer which is responsible for integrating the framework with the resources and services of the testbed. Following a Service Oriented Architecture (SOA), our framework in essence is more an inventory service, rather than a fully functional AM, since functionalities specific to the testbed are not performed by the framework itself. Following the principles of SOA, when a specific task is to be performed on the testbed, the southbound of the framework is responsible for reaching the corresponding service of the testbed. For instance, when an SSH key of a user needs to be uploaded to a server, the framework forwards it to the responsible tool of the testbed, which handles the request and configures the SSH key to the correct account.

The separation of our framework from the testbed's peculiarities, encourages its adoption from similar facilities, since having all functionalities distributed to services offers better control and maintenance of the testbed. On the other hand, maintaining all the functionalities of a complex experimental facility in one software component, is by its nature far more complicated and inefficient. The different service components of a testbed can be reached through the Testbed Adaptation Layer, utilizing both REST and FRCP communication protocols.

V. NITOS Case Study

The NITOS [15] testbed is comprising of heterogeneous type of resources (four different types of wireless nodes, OpenFlow switches, WiMAX and LTE base stations), making this way the management and administration of the facility challenging. Furthermore, NITOS is part of multiple federations with other testbeds adding one more level of complexity for the administrators.

In this section we present the integration of our framework in the NITOS testbed and the challenges we encountered during this endeavour, which provided us with valuable experiences on how to build and manage a wireless testbed. We also give the results from an initial performance evaluation, which helped us to assess the limits of our framework. Following a Service Oriented Architecture, we divided the core functionalities of the NITOS testbed in different service components, incorporating them in our management framework through the adaptation layer described in the previous section.

A fundamental service of NITOS is the Chassis Manager Cards (CMC) service, which is responsible for powering on/off the nodes as well as reseting them, since human intervention is not wanted and is not possible for the remote users of the testbed. The goal is to allow users to issue commands (on/off/reset) to the resources they have reserved, but prohibit unauthorized usage of resources that are not reserved. Thanks to the REST API provided by the management framework, the CMC service can easily discover which resources are reserved and by whom, thus permitting or denying commands received by various users of the testbed.

In the same manner, there is a service responsible for the provisioning of operating system images on the nodes, which contacts first the management framework to verify if a user is allowed to load an image on the requested nodes. This way unauthorized usage of resources can be managed without any intervention in the management framework, which works as an inventory system tracking all the crucial information for the services of a testbed.

An alternative method of service integration is provided through actions/events triggered by the management framework itself. In our case, this approach was carried out for the following services which are responsible for the Linux account management and the OpenFlow resources. More specifically, when a new user/account is created in the management framework through the REST API or the SFA endpoint, an actual Linux account needs to be created in the NITOS server. Since this is something specific to NITOS testbed and there is a service responsible for creating Linux accounts, it should be reached from the management framework for performing the necessary actions. By exploiting the modular design of our framework, the testbed operator needs only to extend a specific module which is responsible for the adaptation layer with the testbed. In turn, the management framework will contact the responsible RC through its FRCP API in order for the latter to create the Linux account.

Similarly, there is an RC responsible for the management of the OpenFlow switches [16]. When a user reserves a number of resources (wireless nodes), the RC sets up their backbone connection with the OpenFlow network by configuring the FlowVisor [17]. To this end, the administrator of NITOS needs to configure the framework to be triggered like in the case of the Linux account creation. The adaptation layer in essence includes all the testbed's specific functionalities which are triggered when a new account is created or when a new reservation is received. An administrator is able to configure the adaptation as he wants based on events that are triggered through the SFA API or when the state of the inventory (resources, accounts) changes.

A. Performance Evaluation

Performance evaluation for software components helps software architects to improve their frameworks. In our system it is critical for the implemented framework to respond as quickly as possible to requests coming from third parties, on both the aforementioned interfaces (REST, SFA).

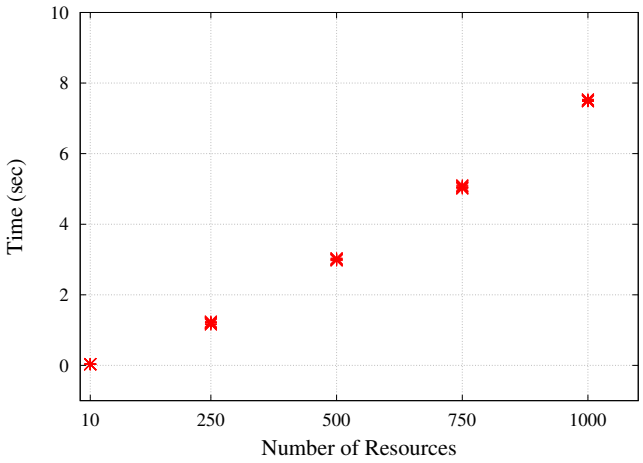

Fig. 2. Performance evaluation of the REST interface for resource advertisements, using a different number of resources.

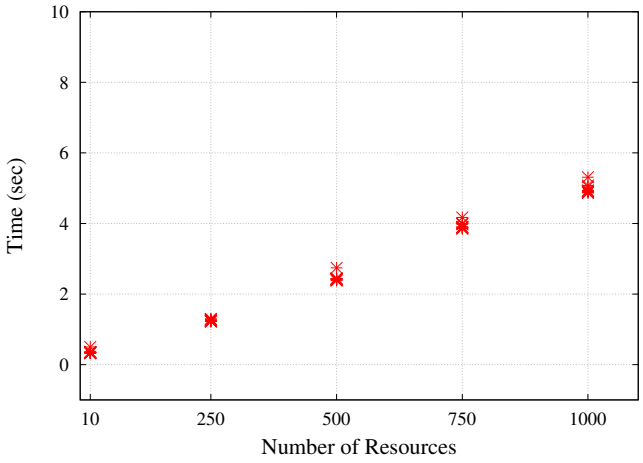

Fig. 3. Performance evaluation of the SFA interface for resource advertisements, using a different number of resources.

The most time consuming procedure of an Aggregate Manager is the advertisement of all resources existing in the inventory. In our experiments we have used five separate inventories with 10, 250, 500, 750 and 1000 resources respectively. For each one of the above inventories we requested 10 consecutive advertisements and measured the total time needed to generate the advertisement individually in the SFA and REST interfaces. We can see the results in Figs. 2 and 3. The REST interface returns a list of ten resources almost in the same time with the SFA interface, while in all the other cases the SFA performs faster than the REST interface. The reason is that in the case of 10 resources, little amount of data is produced by both interfaces, whereas in the other scenarios the difference in the produced data is substantial. It is worth mentioning that the REST interface generates approximately 3 times more data in comparison with SFA (250 bytes compared to 80 bytes of data per advertised node). But the case is that both interfaces are meant to serve different types of requests, thus a one to one comparison is not exactly fair.

REST is used to provide exhaustive resource descriptions in contrast with the SFA which exposes only the basic characteristics of the resources. A significant result obtained from the

```
{
  "name": "node001",
  "urn": "urn:publicid:IDN+omf:nitos.outdoor+node+node001",
  "uuid": "daf96b0a-8a87-469f-bd6b-ed9e480930e8",
  "domain": "omf:nitos.outdoor",
  "available": true,
  "exclusive": true,
  "hardware_type": "PC-Grid",
  "hostname": "node001",
  "ram": "1.8905 GB",
  "ram_type": "DIMM Synchronous",
  "hd_capacity": "59.6263 GB",
  "interfaces": [
    {
      "name": "node001:if0",
      "urn": "urn:publicid:IDN+omf:testserver+interface+node001:if0",
      "uuid": "43a59870-8491-4607-8e5c-73794bf21609",
      "exclusive": true,
      "role": "control",
      "mac": "00:03:1D:0D:90:DE",
      "ips": [
        {
          "uuid": "8e28bf90-a239-4143-9ae0-861e511ddf94",
          "address": "10.0.1.1",
          "netmask": "255.255.255.0",
          "ip_type": "ipv4"
        }
      ],
    },
    {
      "name": "node001:if1",
      "urn": "urn:publicid:IDN+omf:testserver+interface+node001:if1",
      "uuid": "d4f46eaf-b75c-4baf-8172-fe2aea410ea5",
      "exclusive": true,
      "role": "experimental",
      "mac": "00:03:1D:0D:90:DF",
      "ips": [
          ...
      ],
    }
  ],
  "cpus": [
    ...
  ],
  "cmc": {
    ...
  },
}
```

Listing 1. An advertisement of a single resource from the REST interface

performance evaluation is the existence of a tradeoff between the time needed to provide a list of resources and the amount of the information that can be exposed. In order to limit this tradeoff, schemes with different backend technologies should be evaluated like NoSQL databases.

In Listings 1 and 2 we quote examples of resource advertisements for each interface. Due to the extensiveness of output of the REST interface, a limited form of the advertisement is presented. Extra information regarding features like CPU characteristics and chassis manager cards also are being advertised through this interface.

VI. Conclusion and Future Work

Future Internet Infrastructures around the globe are always evolving, offering new and heterogeneous resources over the course of time to the research community. This growth imposes great opportunities for innovations in the context of testbed management frameworks. In this paper we introduced a management framework that is closely related with the OMF control framework and the SFA protocol. We proposed an architecture for testbed Aggregate Managers that separates the actual management of the resources from its core components, introducing an extensible way of supporting versatile type of resources. In addition, we quoted a case study of a complete, real world implementation of the suggested framework, deployed in production mode within the NITOS Future Internet Facility. Last but not least, we executed thorough experiments over our software in an effort to evaluate its performance.

Although the proposed framework is in a stable version, it is also in an ongoing development phase, adding new features

```
<node component_id="urn:publicid:IDN+omf:nitos+node+node001"
  component_manager_id="urn:publicid:IDN+omf:nitos+authority+cm"
  component_name="node001" exclusive="true">
    <available now="true"/>
    <hardware_type name="PC-Grid"/>
    <interface component_id="urn:publicid:IDN+omf:nitos+interface+node001:if0"
    component_name="node001:if0" role="control">
        <ip address="10.0.1.1" ip_type="ipv4"
        netmask="255.255.255.0"/>
    </interface>
    <interface component_id="urn:publicid:IDN+omf:nitos+interface+node001:if1"
    component_name="node001:if1" role="experimental"/>
</node>
```

Listing 2. An advertisment of a single resource from the SFA interface

over the course of time. The implementation of GENI AM version 3 will broaden federation capabilities with third parties by supporting both version 2 and 3 of the SFA protocol. Finally, an idea to extend our architecture enabling an hierarchical structure between different instances of our framework, seems very effective for infrastructures that incorporate more than one testbeds.

VII. Acknowledgements

The authors acknowledge the support of the European Commission through IP project Fed4FIRE (FP7-318389).

References

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

[2] A. Quereilhac, M. Lacage, C. Freire, T. Turletti, and W. Dabbous, "NEPI: An integration framework for network experimentation," in *19th International Conference on Software, Telecommunications and Computer Networks (SoftCOM), 2011 IEEE.*

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

[4] H. Niavis, K. Choumas, G. Iosifidis, T. Korakis, and L. Tassiulas, "Auction-based Scheduling of Wireless Testbed Resources," in *Wireless Communications and Networking Conference (WCNC), 2014 IEEE.*

[5] A.-C. Anadiotis, A. Apostolaras, D. Syrivelis, T. Korakis, L. Tassiulas, L. Rodriguez, and M. Ott, "A new slicing scheme for efficient use of wireless testbeds," in *Proceedings of the 4th ACM international workshop on Experimental evaluation and characterization*, 2009.

[6] Slice-based Facility Architecture. [Online]. Available: http://opensfa.info/doc/opensfa.html

[7] Slice-based Facility Architecture wrapper. [Online]. Available: http://sfawrap.info/

[8] A. Willner and T. Magedanz, "FIRMA: A Future Internet Resource Management Architecture," *Proceedings of the 2014 26th International Teletraffic Congress (ITC)*, 2014.

[9] ORCA Control Framework Architecture. [Online]. Available: https://geni-orca.renci.org/trac/

[10] FOAM OpenFlow aggregate. [Online]. Available: http://groups.geni.net/geni/wiki/OpenFlow/FOAM

[11] 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*, 2014.

[12] OFELIA control framework. [Online]. Available: http://fp7-ofelia.github.io/ocf/

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

[14] GENI Aggregate Manager API Version 3. [Online]. Available: http://groups.geni.net/geni/wiki/GAPI_AM_API_V3

[15] Network Implementation Testbed using Open Source platforms. [Online]. Available: http://goo.gl/j67I6k

[16] D. Giatsios, K. Choumas, D. Syrivelis, T. Korakis, and L. Tassiulas, "Integrating FlowVisor Access Control in a Publicly Available OpenFlow Testbed with Slicing Support," in *Testbeds and Research Infrastructure. Development of Networks and Communities*, vol. 44, 2012.

[17] R. Sherwood, G. Gibb, K.-K. Yap, G. Appenzeller, M. Casado, N. McKeown, and G. Parulkar, "Flowvisor: A network virtualization layer," *OpenFlow Switch Consortium, Tech. Rep*, 2009.