POLITECNICO DI TORINO

DISTRIBUTED PROGRAMMING 2 - Year 2018 / 2019

SPECIAL PROJECT 3

DATA MODELS FOR NFV ARCHITECTURES

Enrico Cecchetti, s253823@studenti.polito.it
Federico Gianno, s255548@studenti.polito.it

Referrals:
Fulvio Valenza,
Jaloliddin Yusupov,
Riccardo Sisto

Contents

1 Introduction

This paper is for the subject *Distributed programming II*. The purpose of the project is to:

- Design a data format (described by means of an XML schema) for the representation of all the most relevant information in the NFV and SDN contexts.
- Design and implement a RESTful web service that permits to store and retrieve the NFV/SDN information.

2 Background

NFV and SDN are emerging paradigms that allow to separate the network's control logic from the underlying routers and switches introducing the ability to program the network. In the following paragraphs, is proposed a standard to manage these two architectures.

2.1 NFV: Network Function Virtualization

The main idea of NFV is the decoupling of physical network equipment from the functions that run on them. This means that a network function, such as a firewall, can be dispatched to a TSP (Telecommunication Service Providers) as an instance of plain software.
The VNFs may then be relocated and instantiated at different network locations without necessarily requiring the purchase and installation of new hardware [1].

2.2 SDN: Software-Defined Networking

SDN is a network paradigm that give a breath of fresh air on the nowadays architecture and that can revolution all the model we are using up to now. The aim of SDN is to provide open interfaces that enable the development of software that can control the connectivity provided by a set of network resources and the flow of network traffic though them, along with possible inspection and modification of traffic that may be performed in the network [2].

3 Starting point

We started from *VerifOO & Verigraph* NFV schema. First of all, we studied its model and compared it with the most successful standard: **ETSI** and **TOSCA**. After summarized the major difference between ETSI and VerifOO/Verigraph, we started to modify their structure in order to be as closer as possible to the standard.
In table 1 are listed the major difference between the two models. Also a resume schema is available here.
To summarize: *VerifOO & Verigraph* is not really compliant with the standard: there are some elements that are missing, other ones with different names and/or attribute, also similar structures sometimes they got additional functional and sometimes less.

VerifOO Verigraph element	ETSI corresponding element	ESTI description	Compliant with the standard
NFV	NSD	NSD consists of statics information elements used by NFV Orchestrator to instantiate a Network Service.	More or less. The purpose is the same but the level of abstraction is different.
Graphs	NSD: nfv_dependency	It defines the sequence in which various nodes and links within a VNF should be instantiated by VNF orchestrator.	ETSI does not provide a specific configuration of that item.
Constraints	NSD:vnfd	It describes a VNF in terms of deployment and operations behaviours requirements.	More or less. It has some matching elements with the standard.
Node constraints	NSD:vnfd:vdu	Information elements concerning the VDU (e.g. processor and memory requirements).	Yes, but the standard is more detailed.
Link constraints	NSD:vld	It describes the basic topology of the connectivity between one or more VNFs, and other required parameters.	Yes, but with less parameters with the respect to the standard.
Property definition	-	-	No, but it's useful for VerifOO/Verigraph workflow.
Hosts	-	-	No, the standard does not provide physical infrastructure implementation.
Connection	-	-	No (same reasons of above).
Network Forwarding Path	NSD:vnffgd	The traffic flow through a VNFFGD is controlled by a Forwarding Path element.	Not so much. In the standard there are a lot of elements that here are missing.
Parsing string	-	-	No.

Table 1: VerifOO/Verigraph vs ETSI

4 Model Design

Network Function Vitalization (NFV) is an entity containing two main blocks:

- The Physical Network Infrastructure (PNI)
- The list of the Network Services offered by the network (NS)

This implementation is not described in the standard, it cares only about the Network Services without infer anything about the physical structure that will host them.
Here, is proposed a different structure with the respect to the standard, because we think that it is worth to store those additional information (PNIs) to manage the allocation of the virtual functions in the physical machines and retrieve them in future.

Identifier	Type	Cardinality	Description
pni	Element	1	Map of the physical network infrastructure.
ns	Element	0...1	List of Network Service Descriptor within the network.

Table 2: Network Function Vitalization

Neither in the *ETSI* standard and *TOSCA* does exist a definition for the NFV. They both start from the NSD entity.
Since PNI is necessary to *VerifOO & Verigraph* has been decided to left the root element (NFV), like in the previous schema, with, in addition to the physical structure, has been defined a new structure containing the Network Services Descriptors (NSD).
The previous schema was composed by both PNI and NSD without a clear difference between them, in terms of attribute and functionality.
To have an overview of the whole schema see this.

4.1 PNI: Physical Network Infrastructure

Entity containing the set of subnets within the network, its hosts and the relative connections between them.

Identifier	Type	Cardinality	Description
hosts	Element	0...N	Hosts which is part of the physical connections.
connections	Element	0...N	Map of the physical connection within the network.

Table 3: NFV:PNI

The whole block contains *Hosts* and *connections* has been moved inside this new entity called **PNI** in order to wrap them together.

We decided to left this block outside the NS because the Physical Network Infrastructure does not concern the description of a Network Service. Then, it should not be included in the NS. This permits also to generalize the schema to make it compatible with wide range of tools.
Moreover, this approach allow us to implement a NS in such way that a future implementation will not require critical changes in the model.

4.1.1 Hosts

A branch (subnet) of the network infrastructure.

Identifier	Type	Cardinality	Description
host	Element	1...N	Physical machine present in the network infrastructure.

Table 4: NFV:PNI:hosts

The previous schema, adopted in *VerifOO & Verigraph*, has been maintained because it is not a bad structure considering the lack of standard design.

4.1.1.1 Host

Identifier	Type	Cardinality	Description
id	Attribute	1	ID of the physical machine.
name	Attribute	0...1	Name of the physical machine.
fixedEndPoint	Attribute	0...1	-
active	Attribute	1	True if at least one node has been deployed on the host.
maxVNF	Attribute	1	Maximum of Virtual Network Function that the host can handle.
type	Element	0...N	Defines the type of the host (e.g. client, server, middle-box).
computational_properties	Element	0...1	Describes the memory and cpu resources characteristics (e.g. cpu, cores, number of operations).
memory_properties	Element	0...1	Represents the physical memory of the host (e.g. memory, disk storage, virtual memory resources).
network_properties	Element	0...1	It represents the network characteristics (e.g. bandwidth).
v_node_ref	Reference	0...N	Reference to a virtual network function.
p_node_ref	Reference	0...1	Reference to a physical network function.
supported_VNF	Element	0...N	Functional types which the host supports.

Table 5: NFV:PNI:hosts:host

The main structure, more or less, remained the same. Some attributes has been grouped based on their purpose (e.g. computational properties: cpu, cores, number of operations, etc.).
Also, has been added a physical node reference in order to connect to the network some physical devices that are not virtualized yet.

4.1.2 Connections

Map of the physical **connection**.

Identifier	Type	Cardinality	Description
connection	Element	0...1	The connection between two or more hosts in terms of source, destination and latency.

Table 6: NFV:PNI:connections

The previous schema, adopted in *VerifOO & Verigraph*, has been maintained because it is not a bad structure considering the lack of standard design.

4.2 NSD: Network Service Descriptor

The NS is composed by a sequence of **NSD**: the Network Service Descriptor is a deployment template for a Network Service refereeing all other descriptors which describe components that are part of that network service.
NSD has been designed has much as possible closer to the standard.
Has been inserted some different entities with the respect to the previous *VerifOO & Verigraph* schema because it was not very compliant with the standard. For the same reason, the name of other elements has been modified with the corresponding ETSI's name.
These elements has been added: id, vendor, version, pnfd, service_deployment_flavour and connection point.
Instead, these elements has been modified:

- Graphs → nfv_dependencies
- Constraints → vnfd
- Network Forwarding Path → vnffgd

4.2.1 VNF Dependency

Describe dependencies between VNF. Defined in terms of source and target VNF, i.e. target VNF "depends on" source VNF. In other words a source VNF shall exist and connect to the service before target VNF can be initiated/deployed and connected. This element would be used, for example, to define the sequence in which various numbered network nodes and links within a VNFFGD should be instantiated by the NFV Orchestrator.

Identifier	Type	Cardinality	Description
id	Attribute	1	ID of this Network Service Descriptor.
vendor	Attribute	0...1	Provider or Vendor of this Network Service.
version	Attribute	0...1	Version of the Network Service Descriptor.
vnf_dependency	Element	0...1	Describes dependencies between VNF.
property_definition	Element	0...1	List of properties that will be checked by VerifOO for a specific graph (useful for VerifOO and Verigraph).
vnf	Element	0...1	VNF which is part of the Network Service.
vnffgd	Element	0...1	VNFFGD which is part of the Network Service.
pnf	Element	0...1	PNFs which are part of the Network Service.
flavours	Element	0...1	Represent the service KPI parameters and its requirement for each deployment flavors of the NS being described. For example flavour describing the requirement for support a service with 300k calls per second.
connetion_points	Element	0...1	List of connection points which acts as an end point of the Network Service.

Table 7: NFV:NS:NSD

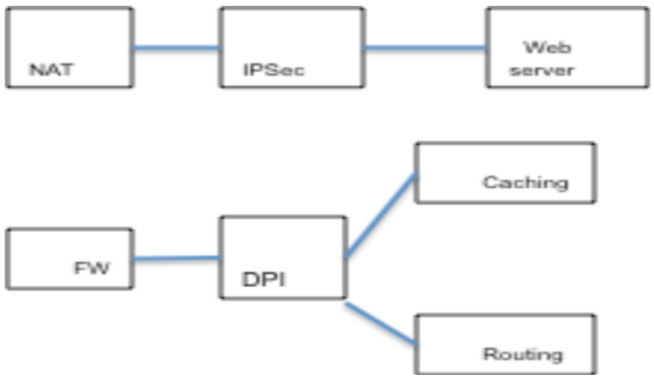

Figure 1: Network composed by two subnets, respectively with 3 and 4 hosts.

The ETSI standard does not provide a specific schema for this element. Therefore, has been decided to not change the main structure proposed by *VerifOO & Verigraph* schema. To be exactly the same as the standard, *graphs* has been renamed as **vnf_dependecy**. Also an attribute has been added: *vnfd_id*, that is a referenced by the *id* of VNFD and it is indicating what kind of VNF that node is providing. See [3], clause 6.2.1.1.

4.2.2 Property Definition

Property Definition is not defined neither in ETSI neither in TOSCA. Has been decided to leave it in order to not compromise the correct functionality of *VerifOO & Verigraph*.

4.2.3 VNFD: Virtual Network Function Descriptor

VNF is a sequence of **VNFD**, a deployment template which describes a VNF in terms of its deployment and operational behavior requirements. The information provided in the VNFD should be used by an Orchestrator to manage and orchestrate Network Services and virtualized resources.

Identifier	Type	Cardinality	Description
id	Attribute	1	ID for this Virtual Network Function Descriptor.
vendor	Attribute	0...1	Provider or Vendor of this Virtual Network Function.
version	Attribute	0...1	Version of the Virtual Network Function Descriptor.
vdu	Element	1...N	Describes a set of elements related to a particular VDU.
virtual_link	Element	0...N	Describes the required parameters. The name has been changed from the previous 'connections'.
dependency	Element	0...N	Dependencies between VDUs. Defined in terms of source and target VDU.

Table 8: NFV:NSD:vnf:vnfd

In the previous model, the **VNFD** entity was ambiguous: it was called with another name (constraints). Its elements *LinkConstraints* and *NodeConstraints*, the actual **vdu** and **virtual_link**, were incomplete and messy. This is due to the fact that the properties was massed all inside to a single entity.
Has been introduced one new entity: *dependency*, present in the standard. Motivation will be reported in the corresponding section.

4.2.3.1 VDU: Virtual Deployment Unit

The **VDU** is a basic part of VNF. It is the VM that hosts the network functions. It describes the properties like *image* to be used in VDU, *management driver* to be used, *flavour* describing physical properties for the VDU to be spawned, etc.

Identifier	Type	Cardinality	Description
id	Attribute	1	ID of this Virtual Deployment Unit.
vm_image	Attribute	0...1	Provides a short description of the VM.
computation_requirements	Element	1	Describes the memory and cpu resources characteristics (e.g. cpu, cores, number of operations).
memory_requirements	Element	0...1	Describes the memory and cpu resources characteristics (e.g. cpu, cores, number of operations).
network_requirements	Element	0...1	Represents the network requirements (e.g. bandwidth).

Table 9: NFV:NS:NSD:vnf:vnfd:vdu

First of all, has been added the attribute **vm_image**, that is a generic description about the characteristic of the virtual machine. It is implemented by both ETSI and TOSCA. However, it has been designed as a generic string because ETSI standard does not provide a specific implementation about the VM information, instead TOSCA propose a very detailed structure about that.
More details about TOSCA VM at [4], see clause 5.4.1.1.
Then, regarding the other changes, *metrics* has been split into three different blocks, each one specific for a requirement type: **computation_requirements**, **memory_requirements** and **network_requirements**. The split is due to the fact that these requirements was all inside the same entity but without a distinction between them. Moreover, this approach will permit future implementation and it is better compliant with the standard.
See [3] clause 6.3.1.2.

4.2.3.2 Virtual Link

Virtual link is a deployment template which describes the resources requirement that are needed for a link between VNFs, PNFs and end-point of the Network Service.

Identifier	Type	Cardinality	Description
src	Attribute	1	Source connection point of a VNF.
dst	Attribute	1	Destination connection point of a VNF.
test_access	Attribute	0...1	Describes test access facilities to be supported on the VL (e.g. none, passive, monitoring or active (intrusive) loopbacks at endpoints).
qos	Element	0...N	Describes Quality of Service options to be supported on virtual link (e.g latency, jitter).

Table 10: NFV:NS:NSD:vnf:vnfd:virtual_link

The structure is more ore less similar to the previous one. The attribute *requiredLatency* has been included inside the new entity **qos** (Quality of Service). The creation of this element permits to include inside a single entity, not only the latency, but also other related parameters. In this way the model is more similar to the standard and is possible to store other data (as the *jitter*) that are cited by ETSI. See [3], clause 6.3.1.3.
Like in TOSCA and ETSI, has been introduced the **test_access** entity. It passively/actively can capture traffic on a network and use it to monitor the network traffic between two points of the network.
See [3], clause 6.3.1.3; or [4], clause 5.9.6.1.

4.2.3.3 Dependency

This a new entity, that represents the constraint in terms of target VDU "depends on" source VDU. In other words, sources VDU shall exists before target VDU can be initiated/deployed.
Speaking about the *xsd* schema, the element **dependency** is a sequence of elements *relations*, each one composed by two references to a source VDU and destination VDU.
See [3], clause 6.3.1.1.

4.2.4 VNFFGD: VNF Forwarding Graph Descriptor

VNFFGD is a deployment template which describes a topology of the network service or a portion of the network service by referencing VNFs and PNFs and Virtual Links that connect them.

Identifier	Type	Cardinality	Description
id	Attribute	0...1	ID for the VNFFG Descriptor.
vnffgd_security	Attribute	0...1	This is a signature of vnffgd to prevent tampering. The particular hash algorithm used to compute the signature, together with the corresponding cryptographic certificate to validate the signature should also be included.
network_forwarding_path	Element	0...N	Describes a network forwarding graph within the VNFFGD.

Table 11: VNF:NS:NSD:vnffgd

The main structure is, more or less, the same as *VerifOO & Verigraph*. Some elements has been renamed: from *Network Forwarding Path* to **vnffgd** and from *path* to **network_forwarding_path**. A new entity has been added **vnffgd_security**, it contains the policy to assure the security of the forwarding path.
See [3], clause 6.5.1.1.

4.2.4.1 Network forwarding path

Identifier	Type	Cardinality	Description
id	Attribute	1	Specify the identifier (e.g. name) of the Network Forwarding Path.
n_endpoint	Attribute	0...1	Count of the external endpoints included in this VNFFG, to form an index.
n_vl	Attribute	0...1	Count of the VLs used by this VNFFG, to form an index.
connection	Element	2...N	Reference to a Connection Point forming the VNFFG.

Table 12: VNF:NS:NSD:vnffgd:netork_forwarding_path

In this section has been added some information that could be useful for some kind of analysis (**n_endpoint** and **n_vl** indexes). Also, the name of *path Node* has been changed to **connection**, as the standard.
See [3], clause 6.5.1.1 & 6.5.1.2.

4.2.5 PNFD: Physical Network Function Descriptor

PNF is a list of**Physical Network Function Descriptor** (PNFD) that describes the connectivity, Interface and KPIs requirements of Virtual Links to an attached Physical Network Function. This is needed if a physical device is incorporated in a Network Service to facilitate network evolution [3].

Identifier	Type	Cardinality	Description
id	Attribute	1	The ID (e.g. name) of this PNFD.
vendor	Attribute	0...1	The vendor generating this PNFD.
version	Attribute	0...1	The version of PNF this PNFD is describing.
description	Attribute	0...1	This element describes an external interface exposed by this PNF enabling connection with a VL.
connection_point	Element	0...1	Physical connection point of the PNFD.

Table 13: VNF:NS:NSD:pnf:pnfd

Has been decided to introduced this new element because it is expected by the standard (see [3], clause 6.6.1). As written in [6], many service providers have made huge investments in these appliance based solutions and quite rightly expect to continue to realize the benefit of these investments for some years into the future.

4.2.5.1 Connection point

The following table represents the structure of the **connection_point** previously cited.

Identifier	Type	Cardinality	Description
id	Attribute	1	ID of the Connection Point
type	Attribute	0...1	This may be for example a virtual port, a virtual NIC address, a physical port, a physical NIC address or the endpoint of an IP VPN enabling network connectivity.

Table 14: NFV:NS:NSD:pnf:pnfd:connection_point

4.2.6 Service Deployment Flavour

TOSCA and ETSI provide two different implementation for the **Deployment Flavour**:

- TOSCA inserts the flavour only inside the Virtual Link Descriptor: it describes a specific flavour of the VL with specific bit-rate requirements.

  See [4], clause 5.9.6.1.

- ETSI propose two different flavour:

– One inside the VNFD, in order to represent the assurance parameters and its requirements for each deployment flavours being descripted (e.g. 10k call per second).
See [3], clause 6.3.1.5.
– One in the NSD, that is the proposed one. It represents the assurance parameters of the Network Service being descripted.
See [3], clause 6.2.1.3.

Identifier	Type	Cardinality	Description
id	Attribute	1	ID of the deployment flavour.
flavour_key	Attribute	1	Assurance parameter against which this flavour is being described. For example, a flavour of a virtual EPC could be described in terms of the assurance parameter "calls per second" (cps).
flavour_value	Attribute	1	Value associated to the flavour_key.

Table 15: NFV:NS:NSD:flavours:service_deployment_flavour

Flavours are constraints about a certain service or function (e.g. 1k calls per second) that must be assured. ETSI propose two levels of flavour: service_deployment_flavour (inside the NSD) and deployment_flavour (inside VNFD).
The flavours contain the constraints and the reference to the VNFs (if NDS's flavour) or VDUs (otherwise) that permits to ensure that.
The idea is that the flavor inside the VNFD could be avoided because it is too hard to design and it carries useless information. Instead, has been implemented the *flavour* of the NSD because it could be useful for future implementation. Right now, it may not be used but in this way the model represents a good starting point for future implementations.

4.2.7 Connection Point

Connection points contains the sequence of **Connection point** that represents a possible connection point of that Network Service.

Identifier	Type	Cardinality	Description
id	Attribute	1	ID of the Connection Point.
type	Attribute	1	This may be for example a virtual port, a virtual NIC address, a physical port, a physical NIC address or the endpoint of an IP VPN enabling network connectivity.

Table 16: NFV:NS:NSD:connectio_points:connection_point

This element has been added because it is provided by ETSI. In the future it could be exploited by connecting different Network Service between them. It may be used in a Software Developed Network.
See [3], clause 6.2.1.2.

5 RESTful web service

The developed **RESTful web service** allows to store and retrieve information about the NFV schema previously described. It permits to add, delete and modify not only the entire structure of the network but also its sub-parts.

5.1 Development

The RESTful has been developed both in Eclipse Neon 2 and IntelliJ IDEA, with the following features:

- Framework: Jersey 2.2 and JAXB
- Server: Tomcat v8.5 (for Eclipse) and Glassfish 4.1 (for IntelliJ)
- Swagger API

The used library are available here.
In the README of the GitHub repo is described how to configure the environment for both *Eclipse* and *IntelliJ IDEA*. Source code is also available.

5.2 Granularity

First of all, it is not allowed to to GET, POST and DELETE the root element **NFV** but is only possible to perform that methods in its sub-root: **PNI** and **NS**. Notice that, for these elements, the POST will overwrite all the previous structures (if there was one). This choice is due to the fact that in the PNI, for example, if you want to add more *Host* or *Connection*, you can just use the relative Method instead of POST from the root. For the same reason, this approach has been adopted also in the NS.
Focusing on the sub-structures of PNI, it is allowed GET, POST, DELETE and also PUT (MODIFY) a single or multiple **Hosts** and/or **Connections**. In this way is very easy to modify the Physical Network Structure. It is not allowed to perform operations on the attributes and/or elements of *Host* and *Connection*: if you want to perform some kind of operations in that attributes/elements just perform it on the Host/Connection. The motivation is that the properties of a certain host will hardly change if the hosts itself will not change. So, at that point, if a specific requirements need to be changed probably also the host would need a change. Therefore, no methods to GET, POST and DELETE elements of Host and Connection.
Regarding **NSD**, it is possible to GET, POST and DELETE a NSD, but is not allowed to PUT (MODIFY) an existing NSD: if you want to modify a certain NSD just use the relative method that will allow you to modify each single element of that NSD.
About **VNFDependency**, same considerations of NSD can be done. Moreover, a POST of VNFDepdency will overwrite all the previous structures. If you want to add something just use the relative method for that element. In this way, for example, if you want to add multiple *graph*, for each graph a response will be provided with a feedback related to the used method. Adding more than one graph contemporaneously, it will not possible to have a response for each graph but it will only possible to have a global response. So, if one graph could not be added to the structure it will not reported

because, maybe, the other n graph will be added correctly. By adding one graph at time it will be possible to have more control on what is happening within the system.
Concerning its elements, **Graph** and **Node**, it is possible to GET, POST and DELETE but not to PUT (MODIFY) a graph. If you want to modify a graph, for sure you need to add, remove or modify some nodes of that graph, so just use the relative method of node that will allow you to POST, DELETE and PUT (MODIFY) a node. If you need to modify the *neighbour* or a *configuration* of a certain node just modify the node. Same motivations of host can be done.
For **PropertyDefinition**, the methods will allow you to GET, POST and DELETE all the defined *property*: for the same reason of VNFDependency, if you POST a *ProeprtyDefinition* it will overwrite all the previous structures. Just use the relative methods of **Property** that will allow you to GET, POST, DELETE and PUT (MODIFY) a single property.
The same considerations of VNFDependency can be done for **VNF**. So, it is permitteed to GET, POST and DELETE of a VNF and the POST will overwrite the previous structures.
Concerning its elements, it is only allowed to GET, POST, DELETE and PUT (MODIFY) the **VNFD**. It is not permitted to perform operations in the elements of that VNFD because of, for example, the **VDU** is describing the property of that VNFD, so if the VDU need to be changed also the VNFD itself need to be changed. Also for the **VirtualLink**, it describes the resources requirements that are needed for a link between VNFs, so if they will change it means that that VNF is not anymore the same or at least it has been changed. Same consideration for **Dependency** that describes the dependencies between VDUs.
Finally, for **VNFFGD**, **PNF**, **Flavours** and **ConnectionPoints** it is allowed to GET, POST, DELETE the root elements and PUT (MODIFY) the related elements. A POST from a root elements will overwrite all the previous structures for the same reason described above.

5.3 Methods

All method listed below are under the path **{project_name}/nfv** and will throw *500 Internal Error* when an unexpected error occurred. More information will follow.

5.3.1 PNI

PNI's methods permits to manage the whole PNI structure.

PNI				
Method	Path	Description	Parameters	Response
GET	/pni	Read the PNI data.	-	200 OK and the PNI structure.
POST	/pni	Add a new PNI. It will overwrite the previous one (if exists).	The body must contain the PNI structure.	201 Created and the added PNI.
DELETE	/pni	Clear the PNI structure.	-	204 No Content.

5.3.1.1 Hosts

Hosts methods permits to manage single or multiple hosts.

Hosts				
Method	Path	Description	Parameters	Response
GET	/pni/hosts	Read all the hosts inside the PNI.	-	200 OK and the Hosts inside the PNI.
POST	/pni/hosts	Add a list of Host.	The body must contain the Hosts structure.	201 Created and the added Hosts.
GET	/pni/hosts/ host/{id}	Read a host information.	The Id of the Host must be in the path.	404 Not Found if that Host does not exist. 200 OK and the requested Host otherwise.
POST	/pni/hosts/host	Add a Host.	The body must contain the Host structure.	403 Forbidden if the Host just exists. 201 Created and the added Host otherwise.
DELETE	/pni/hosts/ host/{id}	Delete an existing Host.	The Id of the Host must be in the path.	404 Not Found if the Host does not exist. 204 No Content otherwise.
PUT	/pni/hosts/host/	Modify an existing Host.	The body must contain the Host structure.	404 Not Found if the Host does not exist. 200 OK and the modified Host otherwise.

5.3.1.2 Connections

Connections methods permits to manage single or multiple connections.

Connections				
Method	Path	Description	Parameters	Response
GET	/pni/ connections	Read all the connections inside the PNI.	-	200 OK and the Connections inside the PNI.
POST	/pni/ connections	Add a list of Connection.	The body must contain the Connections structure.	201 Created and the added Connections.
GET	/pni/ connections/ connection/ {src}&{dst}	Read a connection information.	Source and destination of that connection must be in the path.	404 Not Found if that connection does not exist. 200 OK and the requested connection otherwise.
POST	/pni/ connections/ connection	Add a Connection.	The body must contain the Connection structure.	403 Forbidden if the that connection just exist. 201 Created and the added connection otherwise.
DELETE	/pni/ connections/ connection/ {src}&{dst}	Delete a Connection.	Source and destination of that connection must be in the path.	404 Not Found if the connection does not exist. 204 No Content otherwise.
PUT	/pni/ connections/ connection/	Modify an existing Connection.	The body must contain the Connection structure.	404 Not Found if the connection does not exist. 200 OK and the modified connection otherwise.

5.3.2 NS

NS methods permits to manage the whole NS structure.

NS				
Method	Path	Description	Parameters	Response
GET	/ns	Read the NS data.	-	200 OK and the NS structure.
POST	/ns	Add a list of NSD. It will add the list of NSD and overwrite the previous one.	The body must contain the NS structure.	201 Created and the added NS.
DELETE	/ns	Clear all the NSD.	-	204 No Content.

NSD's methods permits to manage each NSD inside the NS sequence.

NSD				
Method	Path	Description	Parameters	Response
GET	/ns/nsd/{id}	Read the NSD data.	The Id of that NSD must be in the path.	404 Not Found if the NSD does not exist. 200 OK and the NSD otherwise.
POST	/ns/nsd	Add a NSD.	The body must contain the NSD strucutre.	403 Forbidden if the NSD just exists. 201 Created and the added NSD otherwise.
DELETE	/ns/nsd/{id}	Delete a NSD.	The Id of that NSD must be in the path.	404 Not Found if the NSD does not exist. 204 No Content otherwise.

NOTE: The methods below refer each existing NSD.
They have to be called under the path **/ns/nsd/{id}**, where *id* is the identifier of that NSD.
Remember that, entity can be created only if NS and at least one NSD exists. So, each of the method below will work only if the corresponding NSD exists: if you try to GET, POST, DELETE or PUT an element considering a non existing NSD you'll got 404 Not Found (that is referred to the NSD that not exists). More information related to methods response will follow.

5.3.2.1 VNF Dependency

VNF Dependency's methods permits to manage the whole VNF Dependency structure.

VNFDependency				
Method	Path	Description	Parameters	Response
GET	/vnfdependency	Read the VNF Dependency data.	-	200 OK and the VNFDependency structure.
POST	/vnfdependency	Add a VNFDependency. It will overwrite the previous one.	The body must contain the VNFDependency structure.	201 Created and the added VNFDependency.
DELETE	/vnfdependency	Clear VNFDependency.	-	204 No Content.

VNFDependency:Graph-Node				
Method	Path	Description	Parameters	Response
GET	/vnfdependency /graph/{id}	Read a certain Graph data.	The Id of the graph must be in the path.	404 Not Found if the graph does not exist. 200 OK and the Graph otherwise.
POST	/vnfdependency /graph	Add a new Graph.	The body must contain the Graph structure.	403 Forbidden if the graph does exist. 201 Created and the added Graph otherwise .
DELETE	/vnfdependency /graph/{id}	Delete a Graph.	The Id of the Graph must be in the path.	404 Not Found if the Graph does not exist. 204 No Content otherwise.
GET	/vnfdependency /graph/{id} /node/{id}	Read a certain Node of a Graph.	The Id of the Graph and Id of the Node must be in the path.	404 Not Found if the Node does not exist in that Graph. 200 OK and the Node otherwise.
POST	/vnfdependency /graph/{id} /node	Add a new Node in a Graph.	The body must contain the Node structure.	403 Forbidden if the Node just exists in that Graph. 201 Created and the added Node otherwise .
DELETE	/vnfdependency /graph/{id} /node/{id}	Delete a Node of a Graph.	The Id of the Graph and Id of the Node must be in the path.	404 Not Found if the Node does not exist in that Graph. 204 No Content otherwise.
PUT	/vnfdependency /graph/{id} /node	Modify a Node of a Graph.	The Id of the Graph must be in the path.	404 Not Found if the Node does not exist in that Graph. 200 OK and the modified node otherwise.

5.3.2.2 Property Definition

Property Definition's methods permits to manage single or multiple properties.

PropertyDefinition				
Method	Path	Description	Parameters	Response
GET	/propertydefinition	Read all the Properties.	-	200 OK and the PropertyDefinition structure.
POST	/propertydefinition	Add a list of Property Definition. It will overwrite the previous one.	The body must contain the PropertyDefinition structure.	201 Created and the added PropertyDefintion.
DELETE	/propertydefinition	Clear all the PropertyDefintion.	-	204 No Content.
GET	/propertydefinition /property/{id}	Read a Property.	The Id of the Graph (that owns the property) must be in the path.	404 Not Found if that Graph does not have that property. 200 OK and the Property otherwise.
POST	/propertydefinition /property	Add a new Property Definition.	The body must contain the Property structure.	201 Created and the added Property.
DELETE	/propertydefinition /property/{id}	Delete a Property.	The Id of the Graph (that owns the property) must be in the path.	404 Not Found if that Graph does not have that property. 204 No Content otherwise.
PUT	/propertydefinition /property	Modify a Property.	The body must contain the Property structure.	404 Not Found if the Property does not exist. 200 OK and the modified Property otherwise.

5.3.2.3 VNF

VNF's methods permits to manage single or multiple VNFDs.

VNF				
Method	Path	Description	Parameters	Response
GET	/vnf	Read all the VNFDs.	-	200 OK and the VNF structure.
POST	/vnf	Add a list of VNFD. It will add overwrite the previous one.	The body must contain the VNF structure.	201 Created and the added VNF.
DELETE	/vnf	Clear all the VNFD.	-	204 No Content.
GET	/vnf/vnfd/{id}	Read a VNFD.	The Id of the VNFD must be in the path.	404 Not Found if VNFD does not exist. 200 OK and the VNFD otherwise.
POST	/vnf/vnfd	Add a new VNFD.	The body must contain the VNFD structure.	403 Forbidden if that VNFD exists. 201 Created and the added VNFD otherwise.
DELETE	/vnf/vnfd/{id}	Delete a VNFD.	The Id of the VNFD must be in the path.	404 Not Found if the VNFD does not exist. 204 No Content otherwise.
PUT	/vnf/vnfd	Modify a VNFD.	The body must contain the VNFD structure.	404 Not Found if the VNFD does not exist. 200 OK and the modified VNFD otherwise.

5.3.2.4 VNFFGD

VNFFGD's methods permits to manage single or multiple Network Forwarding Path.

VNFFGD				
Method	Path	Description	Parameters	Response
GET	/vnffgd	Read all Network Forwarding Path.	-	200 OK and the VNFFGD structure.
POST	/vnffgd	Add a list of Network Forwarding Path. It will overwrite the previous the one.	The body must contain the Network Forwarding Path structure.	201 Created and the added VNFFGD.
DELETE	/vnffgd	Clear all the Network Forwarding Path.	-	204 No Content.
GET	/vnffgd/nfp/{id}	Read a Network Forwarding Path.	The Id of the Network Forwarding Path must be the path.	404 Not Found if NetworkForwardingPath does not exist. 200 OK and the NetworkForwardingPath otherwise.
POST	/vnffgd/nfp	Add a new NetworkForwardingPath.	The body must contain the Network Forwarding Path.	403 Forbidden if that NetworkForwardingPath exists. 201 Created and the NetworkForwardingPath otherwise.
DELETE	/vnffgd/nfp/{id}	Delete a Network Forwarding Path.	The Id of the NetworkForwardingPath must be in the path.	404 Not Found if the NetworkForwardingPath does not exist. 204 No Content otherwise.
PUT	/vnffgd/nfp	Modify a NetworkForwardingPath.	The body must contain the Network Forwarding Path structure.	404 Not Found if the NetworkForwardingPath does not exist. 200 OK and the modified NetworkForwardingPath otherwise.

5.3.2.5 PNF

PNF's methods permits to manage single or multiple PNFDs.

PNF				
Method	Path	Description	Parameters	Response
GET	/pnf	Read all PNF.	-	200 OK and the PNF structure.
POST	/pnf	Add a list of PNF. It will overwrite the previous the one.	The body must contain the PNF structure.	201 Created and the added PNF.
DELETE	/pnf	Clear all the Physical Network Function.	-	204 No Content.
GET	/pnf/pnfd/{id}	Read a PNFD.	The Id of the PNFD must be in the path.	404 Not Found if PNFD does not exist. 200 OK and the PNFD otherwise.
POST	/pnf/pnfd	Add a new PNFD.	The body must contain the PNFD structure.	403 Forbidden if that PNFD exists. 201 Created and the PNFD otherwise.
DELETE	/pnf/pnfd/{id}	Delete a PNFD.	The Id of the PNFD must be in the path.	404 Not Found if the PNFD does not exist. 204 No Content otherwise.
PUT	/pnf/pnfd	Modify a PNFD.	The body must contain the PNFD structure.	404 Not Found if the PNFD does not exist. 200 OK and the modified PNFD otherwise.

5.3.2.6 Flavours

Flavours methods permits to manage single or multiple Service Deployment Flavours.

Flavours				
Method	Path	Description	Parameters	Response
GET	/flavours	Read all the Service Deployment Service.	-	200 OK and the Flavours structure.
POST	/flavours	Add a list of Service Deployment Service. It will overwrite the previous the one.	The body must contain the Flavours structure.	201 Created and the added Flavours.
DELETE	/flavours	Clear all the Service Deployment Service.	-	204 No Content.
GET	/flavours/flavour/{id}	Read a Service Deployment Service.	The Id of the Service Deployment Service must be in the path.	404 Not Found if Service Deployment Service does not exist. 200 OK and the Service Deployment Service otherwise.
POST	/flavours/flavour	Add a new Service Deployment Service.	The body must contain the Service Deployment Service structure.	403 Forbidden if that Service Deployment Service exists. 201 Created and the Service Deployment Service otherwise.
DELETE	/flavours/flavour/{id}	Delete a Service Deployment Service.	The Id of the Service Deployment Service must be in the path.	404 Not Found if the Service Deployment Service does not exist. 204 No Content otherwise.

5.3.2.7 Connection Points

Connection Points methods permits to manage single or multiple Connection Point.

ConnectionPoints				
Method	Path	Description	Parameters	Response
GET	/cps	Read all the Connection Points.	-	200 OK and the Connection Points structure.
POST	/cps	Add a list of Connection Point. It will overwrite the previous the one.	The body must contain the Connection Points structure.	201 Created and the added Connection Points.
DELETE	/cps	Clear all the Connection Points.	-	204 No Content.
GET	/cps/cp/{id}	Read a Connection Point.	The Id of the Connection Point must be in the path.	404 Not Found if Connection Point does not exist. 200 OK and the Connection Point otherwise.
POST	/cps/cp	Add a new Connection Point.	The body must contain the Connection Point structure.	403 Forbidden Exception if that Connection Point exists. 201 Created and the added Connection Point otherwise.
DELETE	/cps/cp/{id}	Delete a Connection Point.	The Id of the Connection Point must be in the path.	404 Not Found if the Connection Point does not exist. 204 No Content otherwise.

References

[1] Network Function Virtualization: State-of-the-art and Research Challenges - Rashid Mijumbi, Joan Serrat, Juan-Luis Gorricho, Niels Bouten, Filip De Turck, Raouf Boutaba. https://ieeexplore.ieee.org/document/7243304

[2] Software-Defined Networking: A Comprehensive Survey - Diego Kreutz, Member, IEEE, Fernando M. V. Ramos, Member, IEEE, Paulo Verissimo, Fellow, IEEE, Christian Esteve Rothenberg, Member, IEEE, Siamak Azodolmolky, Senior Member, IEEE, and Steve Uhlig, Member, IEEE

[3] ETSI GS NFV-MAN 001 V1.1.1 (2014-12) - Network Functions Virtualisation (NFV); Management and Orchestration - ETSI https://www.etsi.org/deliver/etsi_gs/nfv-man/001_099/001/01.01.01_60/gs_nfv-man001v010101p.pdf

[4] TOSCA Simple Profile for Network Functions Virtualization (NFV) Version 1.0 (Committee Specification Draft 04, 11 May 2017) - OASIS http://docs.oasis-open.org/tosca/tosca-nfv/v1.0/tosca-nfv-v1.0.html

[5] VerifOO RestAPI and XML Dcos - Antonio Varvara, Raffaele Sommese

[6] Physical Network Function (PNF) service chaining with Contrail - OpenContrail http://www.opencontrail.org/physical-network-function-service-chaining-with-contrail/