NFVRG
Internet-Draft
Intended status: Informational
Expires: January 16, 2019

P. Aranda Gutierrez
UC3M
D. Lopez
Telefonica
S. Salsano
Univ. of Rome Tor Vergata/CNIT
E. Batanero
July 15, 2018

High-level VNF Descriptors using NEMO
draft-aranda-nfvrg-recursive-vnf-06

Abstract

Current efforts in the scope of Network Function Virtualisation(NFV) propose YAML-based descriptors for Virtual Network Functions (VNFs) and for their composition in Network Services (NS) These descriptors are human-readable but hardly understandable by humans. On the other hand, there has been an effort proposed to the IETF to define a human-readable (and understandable) representation for networks, known as NEMO. In this draft, we propose a simple extension to NEMO to accommodate VNF Descriptors (VNFDs) in a similar manner as inline assembly is integrated in higher-level programming languages.

This approach enables the creation of recursive VNF forwarding graphs in Service Descriptors, practically making them recursive. An implementation generating VNF Descriptors (VNFDs) for OpenMANO and OSM is available.

Table of Contents

1. Introduction

Currently, there is a lot of on-going activity to deploy NFV in the network. From the point of view of the orchestration, Virtual Network Functions are blocks that are deployed in the infrastructure as independent units. Following the reference architectural model

proposed in [ETSI-NFV-MANO], VNFs provide for one layer of components (VNF components(VNFCs)) below, i.e. a set of VNFCs accessible to a VNF provider can be composed into VNFs. However, there is no simple way to use existing VNFs as components in VNFs with a higher degree of complexity. In addition, Network Service Descriptors (NSD) and VNF Descriptors (VNFDs) specified in [ETSI-NFV-MANO] and used in different open source MANO frameworks are YAML-based files, which despite being human readable, are not easy to understand.

On the other hand, there has been recently an attempt to work on a modelling language for networks or Network Modelling (NEMO) language. This language is human-readable and provides constructs that support recursiveness. In this draft, we propose an addition to NEMO to make it interact with VNFDs supported by a NFV MANO framework. This integration creates a new language for VNFDs that is recursive, allowing VNFs to be created based on the definitions of existing VNFs.

This draft uses two example formats to show how low level descriptors can be imported into NEMO. The first one is the format used in the OpenMANO [1] framework. The second one follows strictly the specifications provided by ETSI NFV ISG in [ETSI-NFV-MANO]. Conceptually, other descriptor formats like TOSCA can also be used at this level.

2. Terminology and abbreviations

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

3. Prior art

3.1. Virtual network function descriptors

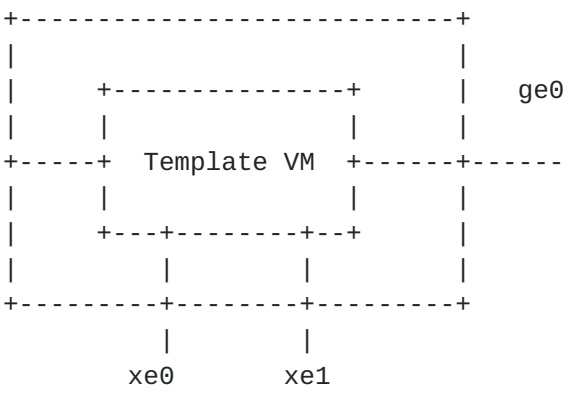
Virtual network function descriptors (VNFDs) are used in the Management and orchestration (MANO) framework of the ETSI NFV to achieve the optimal deployment of virtual network functions (VNFs). The Virtual Infrastructure Manager (VIM) uses this information to place the functions optimally. VNFDs include information of the components of a specific VNF and their interconnection to implement the VNF, in the form of a forwarding graph. In addition to the forwarding graph, the VNFD includes information regarding the interfaces of the VNF. These are then used to connect the VNF to either physical or logical interfaces once it is deployed.

There are different MANO frameworks available. For this draft, we will first concentrate on the example of OpenMANO [2], which uses a

YAML [3] representation similar to the one specified in [ETSI-NFV-MANO]. Then we will provide an example using the exact format specified in [ETSI-NFV-MANO].

3.1.1. OpenMANO VNFDs

Taking the example from the (public) OpenMANO github repository, we can easily identify the virtual interfaces of the sample VNFs in their descriptors:




```
vnf:
    name: TEMPLATE
    description: This is a template to help in the creation of
    # class: parent      # Optional. Used to organize VNFs
    external-connections:
    -   name:              mgmt0
        type:              mgmt
        VNFC:              TEMPLATE-VM
        local_iface_name:  mgmt0
        description:       Management interface
    -   name:              xe0
        type:              data
        VNFC:              TEMPLATE-VM
        local_iface_name:  xe0
        description:       Data interface 1
    -   name:              xe1
        type:              data
        VNFC:              TEMPLATE-VM
        local_iface_name:  xe1
        description:       Data interface 2
    -   name:              ge0
        type:              bridge
        VNFC:              TEMPLATE-VM
        local_iface_name:  ge0
        description:       Bridge interface
```

Figure 1: Sample VNF and descriptor (source: OpenMANO github)

3.1.2. ETSI MANO VNFDs

In this example we consider the VNF represented in Figure 6.4 of [ETSI-NFV-MANO]. Its internal diagram, including a VNF component, is represented in Figure Figure 2. A YAML representation of the VNF Descriptor is reported in Figure Figure 3. The topology of the interconnection of VNFs is expressed by using the abstraction of Virtual Links, which interconnect Connection Points of the VNFs. The

Virtual Links are described by Virtual Link Descriptors (VLD) files. An example YAML representation of the Virtual Link VL1 in the example VNF is reported in Figure Figure 3. In order to understand the topology, a (potentially large) set of VNFD and VLD files needs to be analysed. For a human programmer of the service, this representation is not friendly to write and very hard to read/understand/debug.

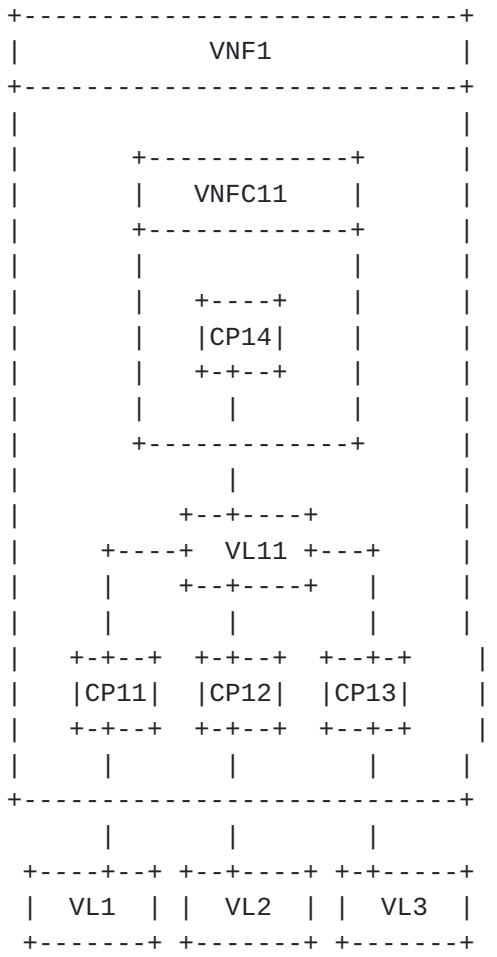


Figure 2: VNF example

```
########################################
# VNF Descriptor of a VNF called vnf1
########################################
id: vnf1
description_version: '0.1'
vendor: netgroup
version: '0.1'
connection_point:
- id: cp11
  type: ''
  virtual_link_reference: vl11
- id: cp12
  type: ''
  virtual_link_reference: vl11
- id: cp13
  type: ''
  virtual_link_reference: vl11
vdu:
- id: vdu11
  computation_requirement: ''
  virtual_memory_resource_element: ''
  virtual_network_bandwidth_resource: ''
  vnfc:
  - id: vnfc11
    connection_point:
    - id: cp14
      type: NIC
      virtual_link_reference: vl11
virtual_link:
- id: vl11
  connection_points_references:
  - cp11
  - cp12
  - cp13
  - cp14
  connectivity_type: ' E-Line'
  root_requirement: ''
```

Figure 3: ETSI MANO compliant VNF descriptor example

```
##########################################
# Virtual Link Descriptor of a VL called vl1
##########################################
id: vl1
descriptor_version: '0.1'
test_access: none
vendor: netgroup
connection:
- cp01
- cp11
connectivity_type: E-LAN
number_of_endpoints: 2
root_requirement: ''
```

Figure 4: ETSI MANO compliant Virtual Link descriptor example

3.2. NEMO

The Network Modeling (NEMO) language is described in [I-D.xia-sdnrg-nemo-language]. It provides a simple way of describing network scenarios. The language is based on a two-stage process. In the first stage, models for nodes, links and other entities are defined. In the second stage, the defined models are instantiated. The NEMO language also allows for behavioural descriptions. A variant of the NEMO language is used in the OpenDaylight NEMO northbound API [4].

NEMO allows to define NodeModels, which are then instantiated in the infrastructure. NodeModels are recursive and can be build with basic node types or with previously defined NodeModels. An example for a script defining a NodeModel is shown below:

```
CREATE NodeModel dmz
  Property string: location-fw, string: location-n2,
    string: ipprefix, string: gatewayip, string: srcip,
    string: subnodes-n2;
  Node fw1
    Type fw
    Property location: location-fw,
      operating-mode: layer3;
...
```

Figure 5: Creating a NodeModel in NEMO

4. Additional requirements on NEMO

In order to integrate VNFDs into NEMO, we need to take into account two specifics of VNFDs, which cannot be expressed in the current language model. Firstly, we need a way to reference the file which holds the VNFD provided by the VNF developer. This will normally be a universal resource identifier (URI). Additionally, we need to make the NEMO model aware of the virtual network interfaces.

4.1. Referencing VNFDs in a NodeModel

As explained in the introduction, in order integrate VNFDs into the NEMO language in the easiest way we need to reference the VNFD as a Universal Resource Identifier (URI) as defined in RFC 3986 [RFC3986]. To this avail, we define a new element in the NodeModel to import the VNFD:

```
CREATE NodeModel <node_model_name> VNFD <vnfd_uri>;
```

4.2. Referencing the network interfaces of a VNF in a NodeModel

As shown in Figure 1, VNFDs include an exhaustive list of interfaces, including the interfaces to the management network. However, since these interfaces may not be significant for specific network scenarios and since interface names in the VNFD may not be adequate in NEMO, we propose to define a new entity, namely the ConnectionPoint, which is included in the node model .

```
CREATE NodeModel <node_model_name>;
  ConnectionPoint <cp_name> at VNFD:<iface_from_vnfd>;
```

4.3. An example

Once these two elements are included in the NEMO language, it is possibly to recursively define NodeModel elements that use VNFDs in the lowest level of recursion. Firstly, we create NodeModels from VNFDs:

```
CREATE NodeModel sample_vnf VNFD https://github.com/nfvlabs
/openmano.git/openmano/vnfs/examples/dataplaneVNF1.yaml;
    ConnectionPoint data_inside at VNFD:ge0;
    ConnectionPoint data_outside at VNFD:ge1;
```

Import from a sample VNFD from the OpenMANO repository

Then we can reuse these NodeModels recursively to create complex NodeModels:

```
CREATE NodeModel complex_vnf;
    Node input_vnf Type sample_vnf;
    Node output_vnf Type shaper_vnf;
    ConnectionPoint input;
    ConnectionPoint output;
    Connection icon Type p2p Endnodes input, input_vnf:data_inside;
    Connection ocon Type p2p Endnodes output, output_vnf:wan;
    Connection intn Type p2p \
        Endnodes input_vnf:data_outside, output_vnf:lan;
```

Create a composed NodeModel

This NodeModel definition creates a composed model linking the sample_vnf created from the VNFD with a hypothetical shaper_vnf defined elsewhere. This definition can be represented graphically as follows:

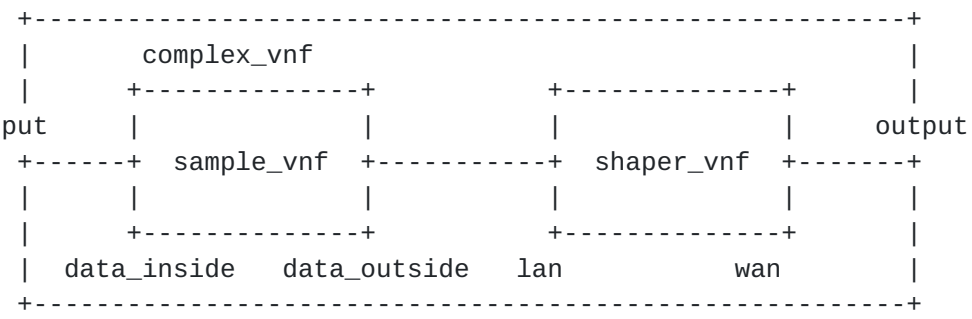


Figure 6

In ETSI NFV, a network service is described by one or more VNFs that are connected through one or more network VNFFGs. This is no more than what is defined in the composed NodeModel shown if Figure 6. By using NEMO, we provide a simple way to define VNF forwarding graphs (VNF-FGs) in network service descriptors in a recursive way.

5. Implementation

There is a proof of concept implementation of the concepts described in this draft is available at github [5]. This proof of concept is implemented as an OpenDayLight (ODL) [6] plugin and includes two output stages to generate VNFDs for OpenMANO and OSM. In its current implementation, the ODL plugin depends on an outdated NEMO project.

This implementation is currently being updated to OpenDaylight Oxygen (the latest version at the time of writing), as a first step towards an ODL-independent implementation.

6. Operational Experience

We have used NEMO descriptors in the context of the MAMI Project [7], to describe a measurement network service based on three virtual network function components:

1. A Trafic [8] traffic generator and sink based on iperf3.

2. A tshark [9]-based packet capture

3. An InfluxDB [10]-based time series database to store measurements

The Network Service Descriptor must always include two instances of the trafic-based VNFC, while the tshak VNFC and the influxdb VNFC are optional (more information is provided at the trafic VNFC creation description page [11].)

The process of creating the different node models is incremental. We start by importing the node models:

```
CREATE NodeModel trafic VNFD https://<repo_url>/trafic.yaml;
    ConnectionPoint mgmt at VNFD:eth0;
    ConnectionPoint gen at VNFD:eth1;

CREATE NodeModel tshark VNFD https://<repo_url>/tshark.yaml;
    ConnectionPoint mgmt at VNFD:eth0;
    ConnectionPoint probe at VNFD:eth1;

CREATE NodeModel influxdb VNFD https://<repo_url>/influxdb.yaml;
    ConnectionPoint mgmt at VNFD:eth0;
```

Figure 7: Creating VNFCs

Then, we create the kernel NSD, based on the trafic VNFCs only:

```
CREATE NodeModel trafic_kernel;
    Node iperf-servers Type trafic;
    Node iperf-clients Type trafic;
    ConnectionPoint client;
    ConnectionPoint server;
    ConnectionPoint mgmt;
    Connection icon Type p2p Endnodes client, iperfs-clients:gen;
    Connection ocon Type p2p Endnodes server, iperfs-servers:gen;
    Connection mgmt Type Lan \
        Endnodes mgmt, iperfs-servers:mgmt, iperfs-clients:mgmt;
```

Figure 8: Kernel NSD based on the trafic VNFCs

Adding the influxdb VNFC to create an autonomous measurement NSD that includes local storage for the measurement results is accomplished with the following NEMO script:

```
CREATE NodeModel
    Node iperf-servers Type trafic_kernel;
    Node database Type influxdb;
    ConnectionPoint client;
    ConnectionPoint server;
    ConnectionPoint mgmt;
    Connection icon Type p2p Endnodes client, trafic_kernel:client;
    Connection ocon Type p2p Endnodes server, trafic_kernel:server;
    Connection mgmt Type Lan \
        Endnodes mgmt, trafic_kernel:mgmt, influxdb:mgmt;
```

Figure 9: Adding influxdb

NEMO has shown a fundamental advantage when compared to YAML or JSON-based descriptors: since it is human-understandable, the development and debugging times of moderate to complex network service descriptors have been shortened considerably and the learning curve is much shallower compared with the original formats.

NEMO allows to identify requirements both for itself and MANO developers more quickly. An example is the connection of the wireshark-based traffic sniffing VNFC. The current connection types (LAN or p2p) do not consider port mirroring, a functionality provided by the TAPaaS plugin in Openstack. This requirement will be fed back to the different MANO communities (OSM, etc.) as a user requirement.

7. Future work

Future work includes extensions to the language to separate control and data plane connections explicitly and new types of connectivity models, including a model that provides the TAP as a Service [12] (TAPaaS) functionality available for OpenStack.

8. Conclusion

With the strategy defined in this document, we are able to link a low-level VNF description into a high-level description language for networks like NEMO. Effectively, we are introducing recursiveness in VNFDs, allowing complex service descriptors to be built by reusing previously tested descriptors graphs as building blocks.

Although we have used the OpenMANO and OSM descriptor formats in this document and for the reference implementation, other descriptors and concepts (i.e. as those used by TOSCA [13]) can also be used as the lowest level in this extension to the NEMO language.

9. IANA Considerations

This draft includes no request to IANA.

10. Security Considerations

The VNFD construct as IMPORT allows referencing external resources. Developers using it in NEMO scripts are advised to verify the source of those external resources, and whenever possible, rely on sources with a verifiable identity through cryptographic methods.

11. Acknowledgement

The work presented in this paper is partially funded by the European Union's Horizon 2020 research and innovation programme under grant agreement No 688421.

12. References

12.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <https://www.rfc-editor.org/info/rfc3986>.

[ETSI-NFV-MANO] ETSI, "Network Functions Virtualisation (NFV); Management and Orchestration", ETSI GS NFV-MAN 001 V1.1.1 (2014-12), December 2014.

12.2. Informative References

[I-D.xia-sdnrg-nemo-language] Xia, Y., Jiang, S., Zhou, T., Hares, S., and Y. Zhang, "NEMO (NEtwork MOdeling) Language", draft-xia-sdnrg-nemo-language-04 (work in progress), April 2016.

12.3. URIs

[1] https://github.com/nfvlabs/openmano

[2] https://github.com/nfvlabs/openmano

[3] yaml.org

[4] https://wiki.opendaylight.org/view/NEMO:Main

[5] https://github.com/telefonicaid/vibnemo

[6] http://www.opendaylight.org

[7] https://mami-project.eu

[8] https://github.com/mami-project/trafic/

[9] https://www.wireshark.org

[10] https://www.influxdata.com

[11] https://github.com/mami-project/trafic/blob/master/README-VM.md

[12] https://docs.openstack.org/developer/dragonflow/specs/tap_as_a_service.html

[13] http://docs.oasis-open.org/tosca/tosca-nfv/v1.0/tosca-nfv-v1.0.html

Authors' Addresses

Pedro A. Aranda Gutierrez
Universidad Carlos III Madrid
Leganes 28911
Spain

Email: paranda@it.uc3m.es

Diego R. Lopez
Telefonica I+D
Zurbaran, 12
Madrid 28010
Spain

Email: diego.r.lopez@telefonica.com

Stefano Salsano
Univ. of Rome Tor Vergata/CNIT
Via del Politecnico, 1
Rome 00133
Italy

Email: stefano.salsano@uniroma2.it

Elena Batanero

Email: elena.batanero.18@gmail.com