ALMA MATER STUDIORUM - UNIVERSITA' DI BOLOGNA

CAMPUS DI CESENA
SCUOLA DI SCIENZE
CORSO DI LAUREA IN INGEGNERIA E SCIENZE INFORMATICHE

# ETSI MANO Network Orchestration

Relazione finale in
Reti di Telecomunicazione

Relatore:
Prof. Franco Callegati

Presentata da:
Riccardo Marchi

Seconda Sessione
Anno Accademico 2017/2018

*To Nicolas*

# Abstract

In the modern era there is a big change in the way computer networks are conceived and the old version defined by hardware implementation is leaving space for a new one based upon software functions. This innovation is the Network Function Virtualization and indeed aims at easing the management of networks and reducing the costs of their maintenance by deploying Virtual Network Functions in standard general purpose servers.

The transition to this solution involved the necessity to improve the performance of virtualization techniques and with the development of new solutions now it is possible to run multiple different functions in the same physical machine. This means that also the cloud computing benefits from this technology, having computing, storaging and networking resources all easily manageable and accessible due to their separation from the hardware underneath.

Therefore it is important that while building this architecture the components are properly working and interacting together and that the virtualization techniques do not produce too much overhead compared to the performance of the hardware implementation.

In this essay will be discussed the Network Function Virtualization and the Open Source MANO project, focusing on its descriptors architecture and functioning. To better demonstrate how to create network topologies through these files, some examples are created and analyzed.

# Contents

# Introduction

The idea of discussing this topic in my thesis came up for two main reasons: the first one is my strongly growing interest in computer networking and its developing design and functioning, while the second one is my intense ambition of always looking to improve and ease life with innovations which everybody can benefit from. My final conviction about dealing with this argument came out of the experience in the mandatory internship during the last year, which made me feel more confident about the future studies I would like to undertake.

I started dealing with a general introduction of the topic and then, step by step, I analyzed a specific platform up until its core components.
The main purpose of this essay is to give a complete explanation of how network topologies are made using YAML files according to ETSI MANO standard and to try to analyze how those lines of code are interpreted by OSM platform.

Chapter 1 concerns the birth of the Network Function Virtualization project and an exhaustive description of the architectural framework is provided. A brief explanation of what is a Network Service completes the section.

Chapter 2 puts the focus on the Management and Orchestration section of the architecture and introduce an open source platform (OSM) specifying its development features and its framework structure.

Chapter 3 coincides with the core of the essay, the part where the descriptor files are illustrated and clarified the most in detail possible and first examples are reported and explained.

Chapter 4 represents an attempt to create network topologies in OSM platform and, without having tested them, it is provided an explanation of why it is supposed to work taking a closer look to the key lines of code produced.

Chapter 5 draws the conclusions and exposes the author's point of view about the study and work accomplished.

# 1. Network Function Virtualization

## 1.1 - Introduction to NFV

In 2012 the European Telecommunications Standards Institute (ETSI) founded an Industry Specification Group (ISG) to develop a standardized approach to Network Function Virtualization (NFV).
This group is the union of 7 Telco operators and until nowadays it counts more than three hundred companies.
Currently, Network Providers rely on proprietary components to provide their services around the internet, implying that each time a new service has to be deployed or a new feature has to be added, the company needs to perform a reconfiguration and usually in-place installation of new hardware too. Therefore this requires further space, electrical power and trained maintenance staff, meaning additional costs for the company.
Knowing that each vendor designs his own function boxes with no interoperability guarantees with other vendor's appliances, NFV goal is to make networks agile and capable of responding automatically to the needs of the traffic and services running on top of it, in the same way applications are supported by dynamically configurable and fully automated cloud environments.
In order to facilitate the development of NFV components across different open source projects, in 2014 the Linux Foundation announced the release of a new open source reference platform with the cooperation of ETSI, the Open Platform for Network Function Virtualization (OPNFV), perfectly matching the ETSI architecture in each of its components.
At the beginning it only concerned the implementation of the Network Function Virtualization Infrastructure (NFVI) and the Virtual Infrastructure Machine (VIM) parts, but later it involved MANO section too.

ETSI is also hosting another MANO initiative, OSM, an operator-led community that aims at delivering a working open source MANO stack aligned with ETSI NFV Information Models and that meets the requirements of NFV networks.
NFV targets are not fixed networks components only, but the same amount of attention is given to mobile networks ones too.

## 1.2 - NFV Architectural Framework

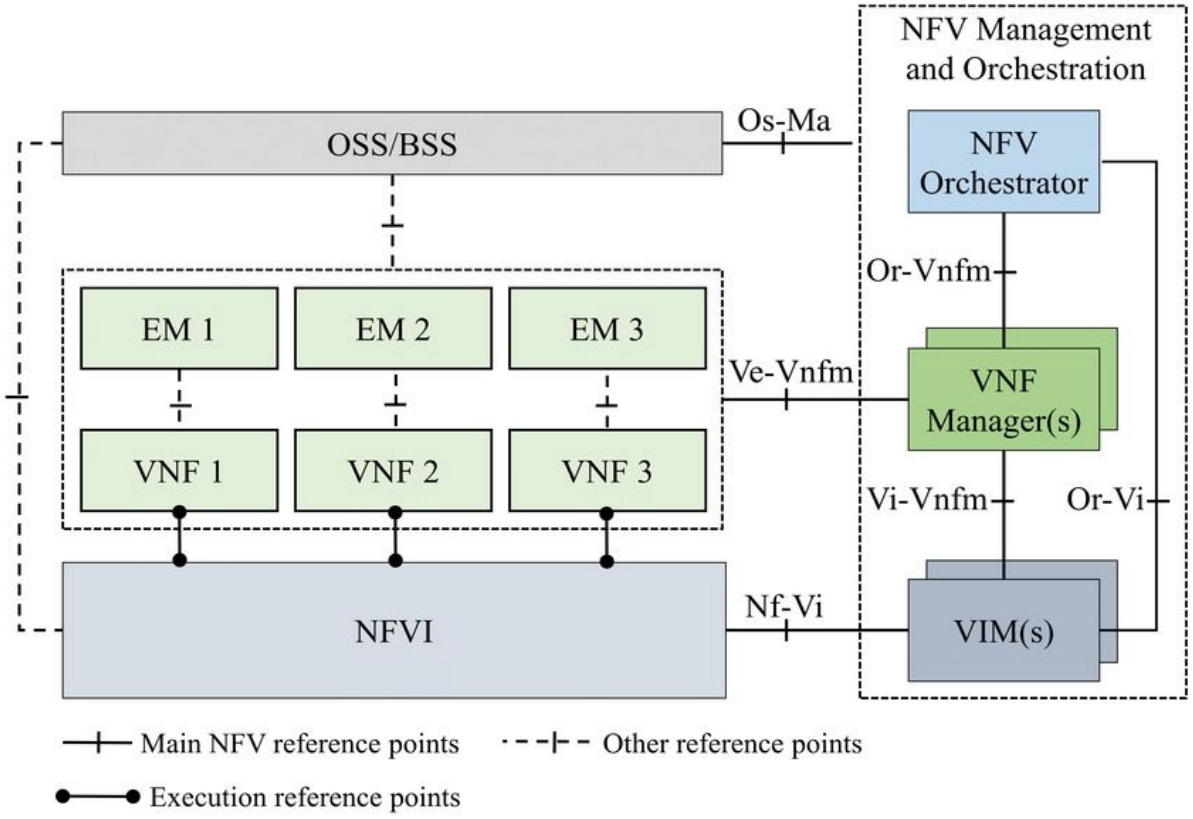


Figure 1.1: ETSI NFV framework architecture.

Figure 1.1 represents the complete architectural framework defined by ETSI. Each macro-component will now be discussed in detail.

## 1.2.1 - NFV Infrastructure

The Network Function Virtualization Infrastructure (NFVI) consist of the hardware and software components where Virtual Network Functions are deployed.
Their goal is to create a virtualization layer to abstract the underlying hardware resources, which will be logically partitioned and then provided to the Virtual Network Functions (VNF) to perform their duties.
Inside this component, ETSI distinguishes three different domains, shown in Figure 1.2.

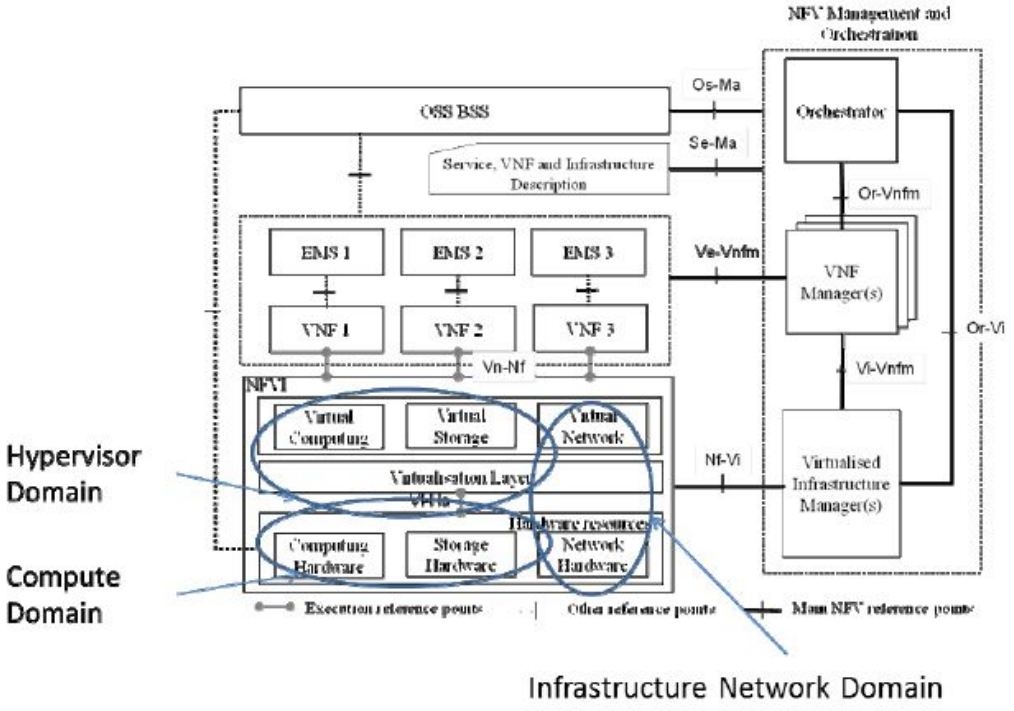


Figure 1.2: NFVI domains.

**Compute Domain**: This domain contains all physical resources available as commercial off-the-shelf. Network connectivity is provided through interfaces to the network infrastructure domain, while the resources provided are abstracted into logical resources by a virtualization layer. This is an example of how NFV could influence the development of virtualization and Cloud technologies.

**Hypervisor Domain**: The hypervisor domain tasks are converting physical resources into logical ones and providing them to those Virtual Machines (VM) running on top of them. There are many ways to perform abstraction, but the most suitable one must be chosen by referring to the specific needs: if there are some latency requirements there may be needed a lightweight virtualization technique, while other applications may require a specific Instruction Set Architecture (ISA) and therefore would be necessary a full emulation of a different ISA.

**Infrastructure Network Domain**: The infrastructure network domain comprises all the communication channels for the connectivity through different components. It provides communication between Virtual Network Function Components (VNFC) of a VNF, between different VNFs. In addition, it guarantees communication between VNFs and the Network Functions Virtualization Orchestrator (NFVO), and between NFVI and NFVO.

### 1.2.2 - NFV Management and Orchestration (MANO)

While Figure 1.1 shows on the right side the three main blocks of the component responsible for network management and orchestration, Figure 1.3 represents a more detailed view of each element.
The decoupling of a Virtual Network Function (VNF) from the underlying hardware resources presents new management challenges. Such challenges include end-to-end service to end-to-end NFV network mapping, instantiating VNFs at appropriate locations to realize the intended service, allocating and scaling hardware resources to the VNFs, keeping track of VNF instances location, etc. Such decoupling also presents challenges in determining faults and correlating them for a successful recovery over the network. These are all tasks MANO needs to accomplish the best way possible.

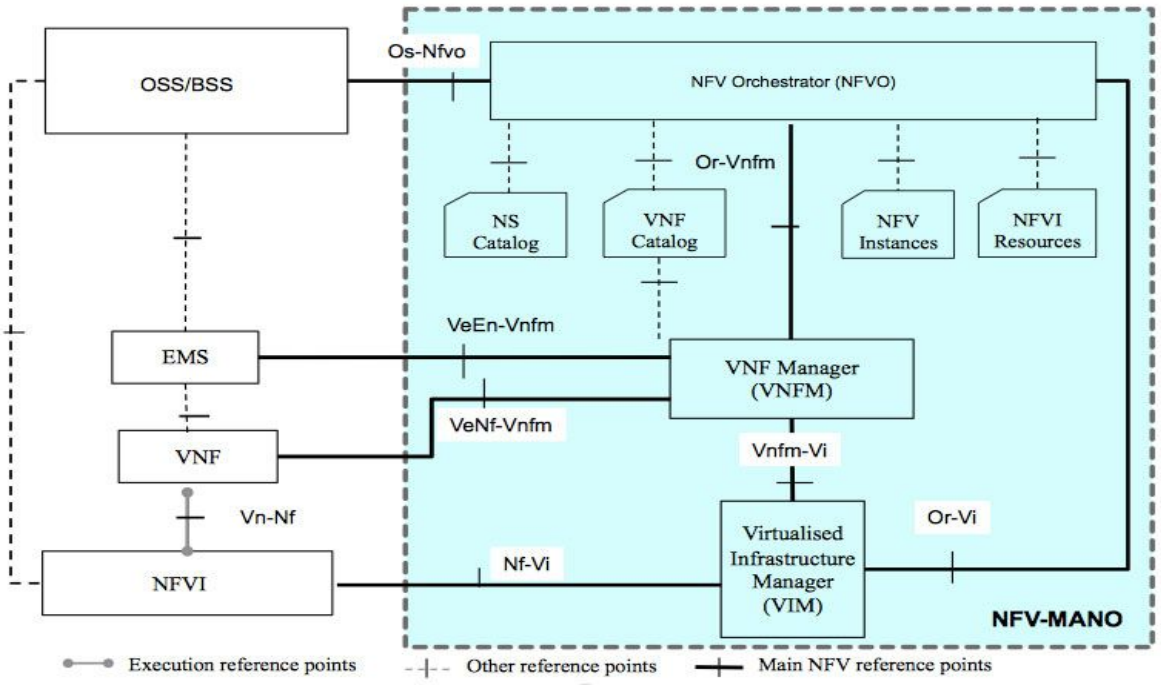


Figure 1.3: ETSI MANO Framework

This separation of elements leads us to the introduction of new entities:

- Network Service (NS): Composition of network functions in order to execute a certain task.
- Virtual Network Function (VNF): Software implementation of a physical network function.
- Physical Network Function (PNF): Legacy hardware implementation of a network function. It is crucial to guarantee the coexistence between PNF and VNF, in order to have a seamless shift from one solution to the other.
- Virtual Network Function Component (VNFC): Basic software component that in addition to others creates a VNF.
- Network connection: It is defined by connection points and Virtual Links (VL). A connection point is the software implementation of a network interface card. A VL is the software implementation of a physical connection (i.e. a virtual cable). A VL may be used to connect both VNFs and PNFs. Two types of VLs are defined: An external VL is used either to connect a NS to the outside world or to connect different VNFs within the same NS. An internal

VL is used either to connect different VNFCs within the same VNF or to connect VNFC to the external interface of a VNF.

- Virtual Network Function Forwarding Graph (VNFFG): Graph made of logical links connecting network function nodes. Essentially it represents the flow of the packets through the functions composing the NS.

As we can easily distinguish in Figure 1.1, the three main blocks of MANO are the Virtualized Infrastructure Manager (VIM), the Virtual Network Functions Manager (VNFM) and the NFV Orchestrator (NFVO).

**Virtualized Infrastructure Manager (VIM)**: This component is responsible for the management of NFVI and through hypervisors or network controllers it is able to control computation, storage and network resources assignment. The main job it is asked to do is to keep trace of all the virtual resources available and also those which have been allocated to physical ones (due to the ability of the VIM to exchange information with more than one machine simultaneously). In addition to that, depending on the type of implementation, each VIM could have more functionalities, that could be the management of security groups policies to ensure access control or gathering performance and fault information.

**Virtual Network Functions Manager (VNFM)**: This element has the task of managing the lifecycle of the VNF, which means starting with the instantiation (and configuration), passing through its management and finishing with the termination. Obviously, in order to perform all these actions it is not sufficient having only the software running a specific VNF, but it is necessary an information model that uses descriptors (defined by ETSI), which will determine the deployment and the operational behaviour of each VNF. The ETSI MANO information model will be discussed in another chapter later.
The VNFM is not only connected to the VNFs, but to the Element Management Systems (EM/EMS) too, which are responsible to perform FCAPS(Fault Management, Configuration Management, Accounting Management, Performance Management, Security Management) for their respective VNF.

While VNFM takes care of the VNF virtual part (everything regarding their lifecycle), on the other hand EMS is focused on the functional part (how to perform its actions).

**NFV Orchestrator (NFVO)**: This block has two tasks, which are the Resource Orchestration and the Network Service Orchestration.
The former functionality is used to provide services to access NFVI in an abstract way without caring about the information regarding any VIM. The latter has the job to orchestrate the NS instantiated, by coordinating the multiple VNFs that compose each one; to do that, it uses the services exposed by the VNFM and the Resource Orchestrator.
This block is connected to the Operation Support System (OSS)/Business Support System (BSS) of the server provider. Although network operators make huge investment in the optimization of OSS/BSS for their own environment, they will need to evolve their one in order to support a NFV-based architecture, that is characterized by highly dynamic network changes. Indeed, in a NFV context, the network architecture, topology and service delivery chain can change very frequently. Moreover, considering that service delivery has to support a multi-vendor environment, service and application monitoring becomes challenging. Legacy OSS/BSS solutions are made of a set of interconnected applications each focusing on specific functions. The high dynamism introduced by NFV requires real-time monitoring and full automation, that is not supported by most of currently deployed OSS/BSS. The development of open source OSS/BSS solutions and guidelines would help operators to perform this shift.

## 1.3 - NFV Network Service

As briefly explained before, a Network Service is a composition of different Network Functions. In Figure 1.4 it is possible to see that a NS can present multiple VNF

Forwarding Paths (VNFFP), which are the perfect expression of "Function Chaining". The primary advantage of this innovation is to mechanize the way virtual network connections can be set up to handle traffic flows for connected services. It also can be operationally beneficial by enabling automated provisioning of network applications that may have different characteristics. For example, a video or VOIP session has more demands than simple Web access. Automated function chaining can enable these sessions to be set up and torn down dynamically, without requiring human intervention. This also helps ensure that specific applications are getting the proper network resources or characteristics (bandwidth, encryption, quality-of-service).

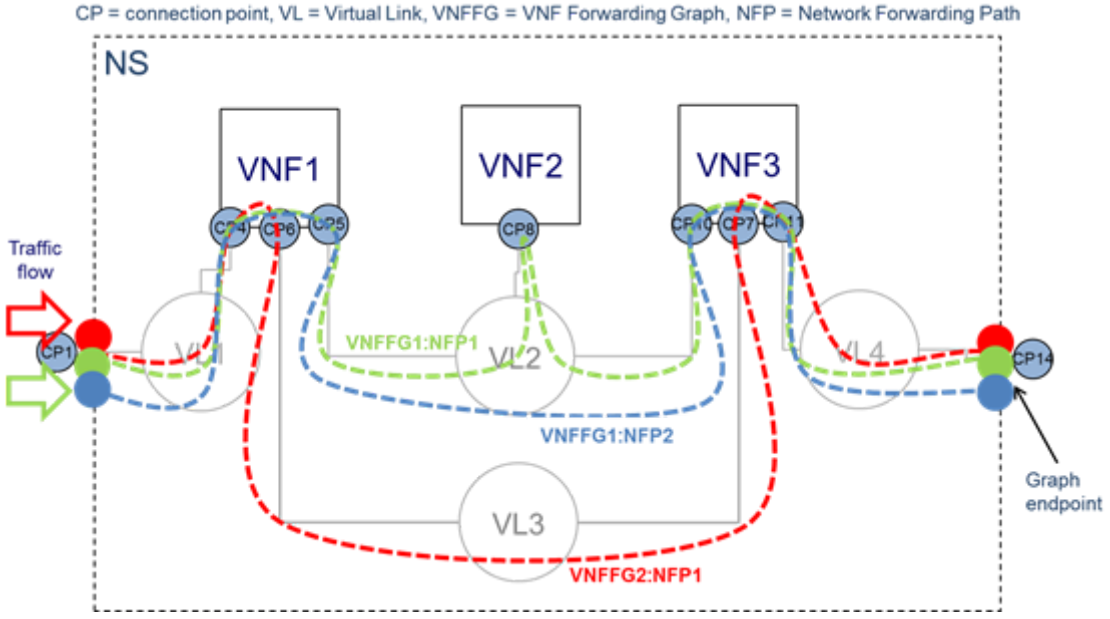


Figure 1.4: NS example with different forwarding paths.

Each VNF has one or more Connection Point (CP), the software implementation of a Network Interface Card. The composition of VLs and CPs define a Network Forwarding Path (NFP), in which the traffic could flow from one end point to another exploiting the connected VNFs.

# 2. Open Source Mano

Open Source Mano (OSM) is an ETSI-hosted open source management and orchestration (MANO) community project to jointly innovate, create and deliver a MANO stack that is closely aligned with ETSI NFV information models and that meets the requirements of commercial NFV networks.
The first release of the ETSI MANO platform was announced during Mobile World Congress (MWC) in 2016 and was issued in June of the same year as Release 0. The operators leading the development group of the platform decided to issue a release every 6 months and right on schedule Release 4 has been announced on May 2018.

## 2.1 - Features

Figure 2.1 shows the mapping of OSM components within the NFV MANO architectural framework.
The main feature of this open source project is to design it in order to be interoperable with components developed by different vendors. In fact, as we can see in Figure 2.1, it is able to support a notable group of VIM implementations, like AWS, OpenStack, OpenVim and VMware. Additionally it is able to manage multiple Software Defined Network (SDN) controllers, like Open DayLight (ODL), Open Network Operating System (ONOS) and Floodlight.

Talking about SDN technology, it is is an emerging architecture that is dynamic, manageable, cost-effective, and adaptable, making it ideal for the high-bandwidth, dynamic nature of today's applications. This architecture decouples the network control and forwarding functions enabling the network control to become directly programmable and the underlying infrastructure to be abstracted for applications

and network services. This innovation is strictly connected to NFV, but it is not going to be discussed in this essay.

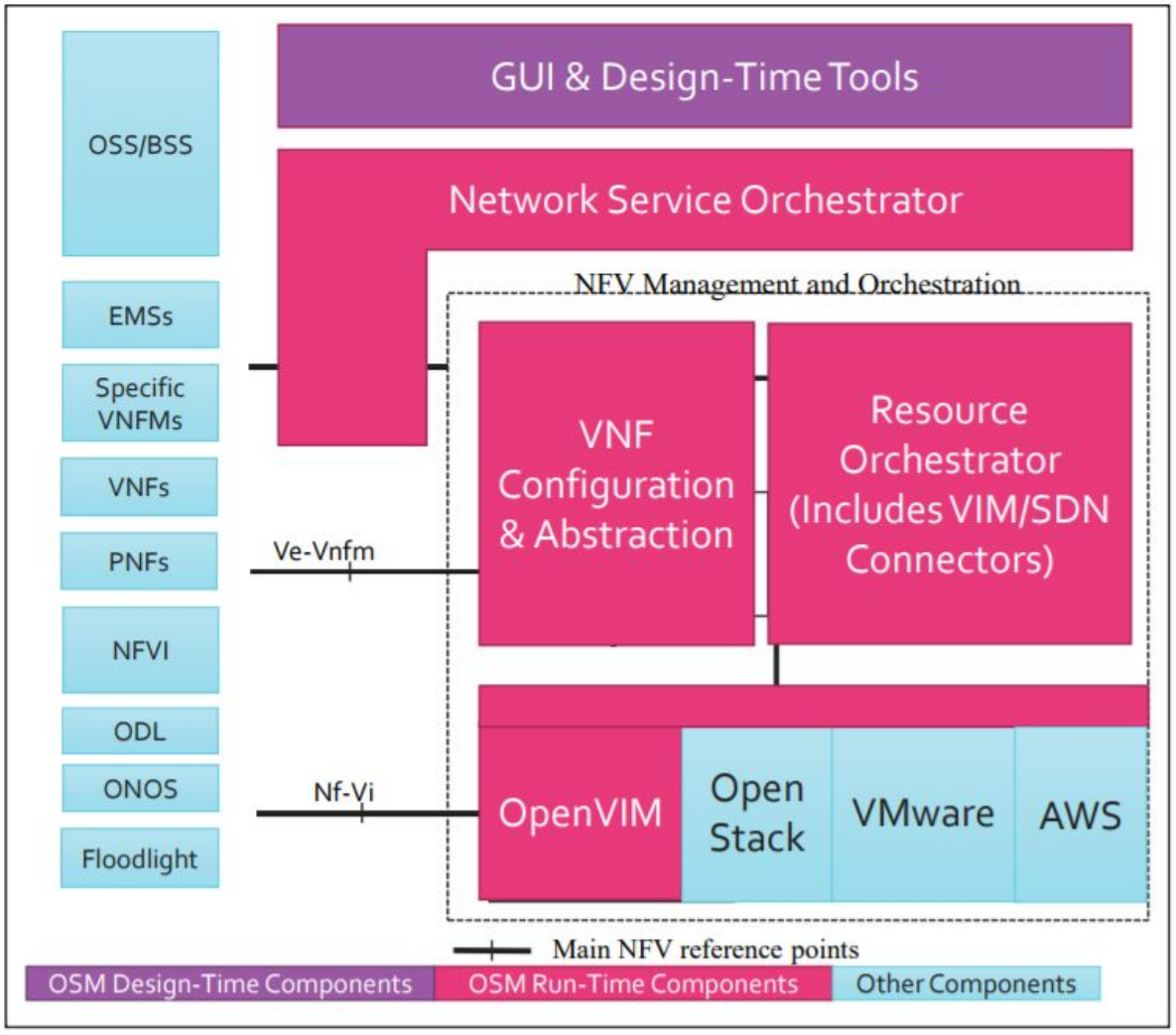


Figure 2.1: OSM components mapped on ETSI NFV MANO architectural framework

With the second release (Release 1) the multi-site capability is added to the platform, usefully letting the instantiation of a NS made of multiple VNFs dislocated geographically in different data centers.

To maintain the near-native performances of the virtualized environment, a new innovation was required, also in order to make all the system work in the proper way in real deployment scenarios. Therefore OSM developed the Enhanced Platform Awareness (EPA), which actually is only supported by a restricted group of VIMs. By using this platform, a Cloud OS is able to understand where a VM can perform better

in a certain situation. In fact the VIM has the capability to discover all by itself the enhanced features in the CPU and PCIe slots available on the servers it manages, choosing the better one for each instantiated VM.
Another important feature added (included in Release 3) is the Role Based Access Control (RBAC), which is a widespread technique adopted to restrict system access to authorized people only. It consists of a different set of permissions for each group, in which every user has those limited visibility and control authorizations.
Furthermore, Release 4 made the platform much easier to install and operate, increasing its interoperability and scalability. Architectural improvements resulted in a more efficient behaviour, reducing the RAM consumption up to 75%.

In this essay it will be examined the platform up until Release 3, the latest available when the study of the platform started.

## 2.2 - Architecture

Figure 2.2 is a detailed representation of OSM logical architecture. There are three main components on which the entire platform is built on, guaranteeing a certain level of abstraction without being too excessive. Those pre-existent elements are the Resource Orchestrator OpenMano, delivered by Telefonica, the Service Orchestrator Riftware and the VNF Manager Juju. It is good to know that in OSM environment the VNFM is often referenced as Configuration Module or Virtual Network Functions Configuration and Abstraction (VCA) module.
By reducing the level of abstraction using components coming from different vendors, involves a non-ignorable problem of inconsistency due to the several implementation methods, naming and file location. To resolve this issue the DevOps module, on the left of Figure 2.2, has been built and helps to have a continuous and unidirectional workflow in order to ease developer's job and to speed up the releasing process.
Since it is not an essential part of the platform it will not be examined in this essay.

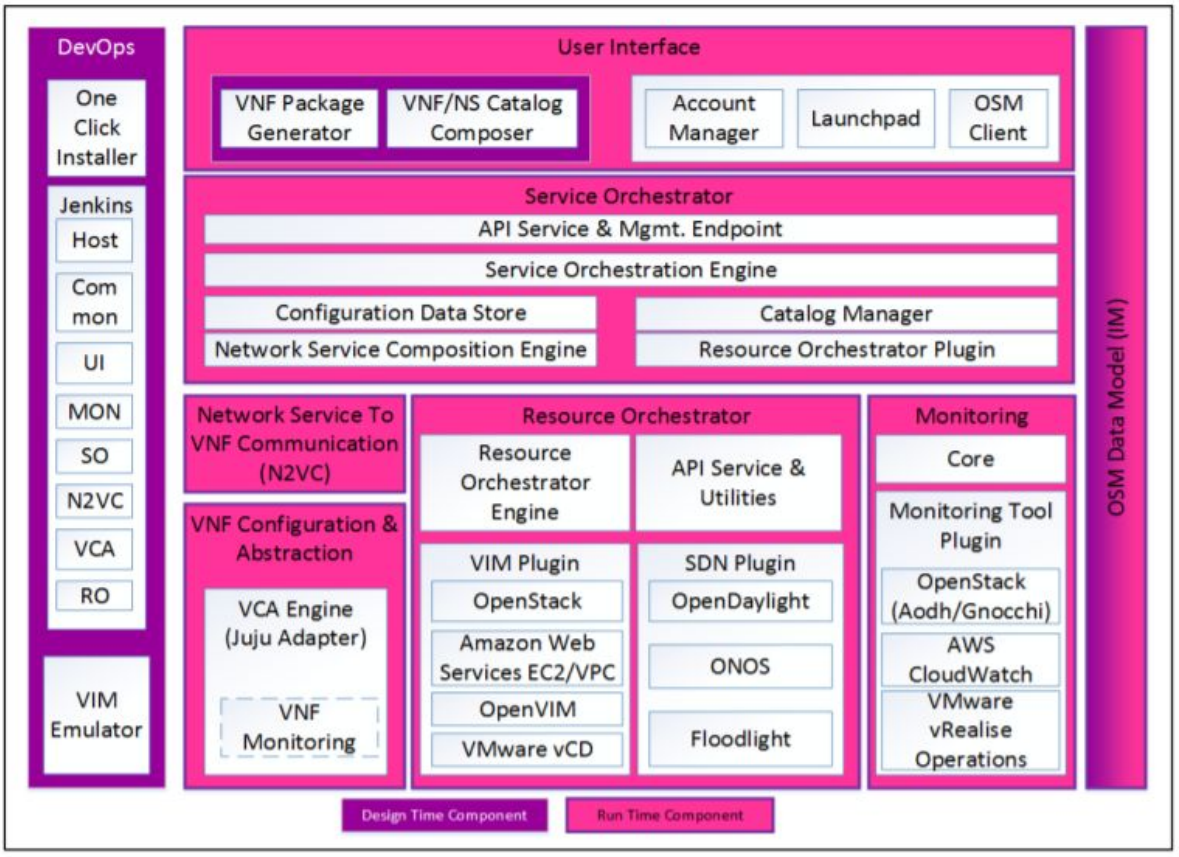


Figure 2.2: Detailed view of OSM logical architecture.

## 2.2.1 - User Interface

The first component of OSM logical architecture is the User Interface (UI) Module, made by both design and run time elements.

The design-time parts are tools used to build properly-formed packages. These packages are fundamental to the creation of the services, for the fact that they are the main carriers of information regarding the functions and networks we would like to instantiate. In fact these packages are composed by a checksum file, a README file and a descriptor file. Many other items can be included in these packages, depending on the needs of the service.
The two tools responsible for these actions are the VNF Package Generator and the VNF/NS Catalog Composer.

The Package Generator aims to create a well-formed package given the descriptor file. The available utilities are a Command Line Interface (CLI) or a web based-form (provided by riftware), with the benefit to be very easy to use if the descriptors are not too complex. Otherwise it would be inconvenient and occasionally troublesome.

The Catalog Composer, instead, is an helpful tool to create the desired descriptor file by filling some information in the Graphical User Interface (GUI) provided by OSM. Moreover it allows you to add and configure the component needed according to the newly created topology. An additional button is provided to let the user see the yaml format of the desired descriptor.

The run-time parts are the Account Manager, the Launchpad and the OSM Client.

The Account Manager is responsible for the configuration and management of the credentials of each part of the platform, regulating the access for every account. It is important to notice that, for example, there are different accounts in the RO in comparison to VIM's ones.

The Launchpad is the interactive GUI with which it is possible to manage lifecycle operations like instantiating or turning down any type of service. It is also useful because it allows you to see real time statistics and a detailed view of the network topologies created.

The OSM Client is a simple CLI using a python functional library that allows users to interact with the system remotely.

### 2.2.2 - Service Orchestrator

Every element belonging to the Service Orchestrator Module is a run-time component.

The API Service & Management Endpoint is providing the primary API endpoint into OSM.

The Service Orchestration Engine is responsible for all aspects of service orchestration including lifecycle management and service primitive execution. It is effectively the “master” orchestration component in the system that governs the workflow throughout OSM. It is also responsible for supporting the concepts of multi-tenancy, projects, users, and enforcing role-based access controls.

The Configuration Data Store is responsible for persistently storing the SO state, particularly in the context of VNF and NS descriptors deployment records.

The Network Service Composition Engine is responsible for supporting NS and VNF descriptors composition. It validates that these files conform to the defined YANG (Yet Another Next Generation, data modeling language) schema.

The Catalog Manager is responsible for supporting the Create/Read/Update/Delete lifecycle operations on the defined VNF and NS descriptors and packages.

The Resource Orchestrator Plugin is responsible for providing an interface to integrate the Resource Orchestrator.

### 2.2.3 - NS to VNF Communication

The NS to VNF Communication (N2VC) Module is responsible for the plugin framework between the SO and the VNF Configuration and Abstraction (VCA) layer.

### 2.2.4 - VNF Configuration & Abstraction

The VNF Configuration and Abstraction (VCA) layer is responsible for enabling configurations, actions and notifications to/from the VNFs and Element Managers.

### 2.2.5 - Resource Orchestrator

The API Service & Utilities endpoint is responsible for providing the interface into the RO (for the SO to consume) and has available a number of utilities for internal to RO consumption.

The Resource Orchestration Engine is responsible for managing and coordinating resource allocations across multiple geo-distributed VIMs and multiple SDN controllers.

The VIM and SDN Plugins are responsible for connecting the Resource Orchestration Engine with the specific interface provided by the VIMs and SDN controllers.

### 2.2.6 - Monitoring

One of the guiding principles for the OSM Monitoring Module (MON) is that it is required to interface with and leverage existing or new monitoring systems. It is not intended to replicate or compete with those ones.
The Monitoring Module should mostly be considered as a means for driving configuration updates to the external monitoring tools and as a conduit for steering actionable events into the Service Orchestrator. These actionable events may be either directly triggered by running NS/VNFs or deduced by the external monitoring tools. One of the most powerful things OSM is delivering as a part of the Monitoring Module is the ability to correlate telemetry related to the VMs and VNFs to the relevant NSs.
Automated correlation is expected to provide a considerable user experience improvement to OSM users and drive up efficiency for operators in a Telecommunications environment.

### 2.2.7 - OSM Information Model

OSM is based on a model-driven architecture. The architectural direction has always been to use the same model as the basis of both the design-time capabilities and the run-time capabilities. The OSM Information Model Module was created to be the single point of authority on the OSM data model that is leveraged by the different components. This helps the development towards a methodology where two of the most important data models in the system, the VNF Descriptor (VNFD) and the Network Service Descriptor (NSD), can be shared in their innate forms between components. OSM modules can act authoritatively on the relevant parts of the VNFD/NSD.

# 3. Descriptor Files

One of the main goals of NFV innovation is to establish a standard way to create, manage and update virtual networks. Therefore the first step of this development is to make different pieces of hardware co-work together without creating any trouble. Once completed this job, as happened with OSM components (dealt in the previous chapter), the following step is to standardize the software language in order to make development and maintenance of the platform easy for multiple programmers coming from different companies. So the key element to address is the data modeling language, which would define functions tasks and networks topologies.
The selected schema for this purpose is the Yet Another Next Generation (YANG), a data modeling language used to design configuration and state information manipulated by the Network Configuration (NETCONF) Protocol, NETCONF remote procedure calls, and NETCONF notifications. It can be converted into any encoding format (like XML or JSON) and would still be supported by the standard. As it will be better analyzed later in the chapter, OSM uses YAML files for data modeling, a supported superset of JSON but more human readable and, by using references, can also be more efficient.

## 3.1 - MANO Descriptor Overview

The information model defined by ETSI uses descriptors, configuration templates that define the main properties of managed objects in a network. Depending on the type of the component, they will define their deployment and operational behaviour.

Figure 3.1 illustrates the structure of a MANO NS, showing which descriptor files are needed to instantiate a service and all its elements: Network Service Descriptor (NSD), Virtual Network Function Descriptor (VNFD), Physical Network Function Descriptor (PNFD), Virtual Link Descriptor (VLD) and VNF Forwarding Graph Descriptor (VNFFGD).

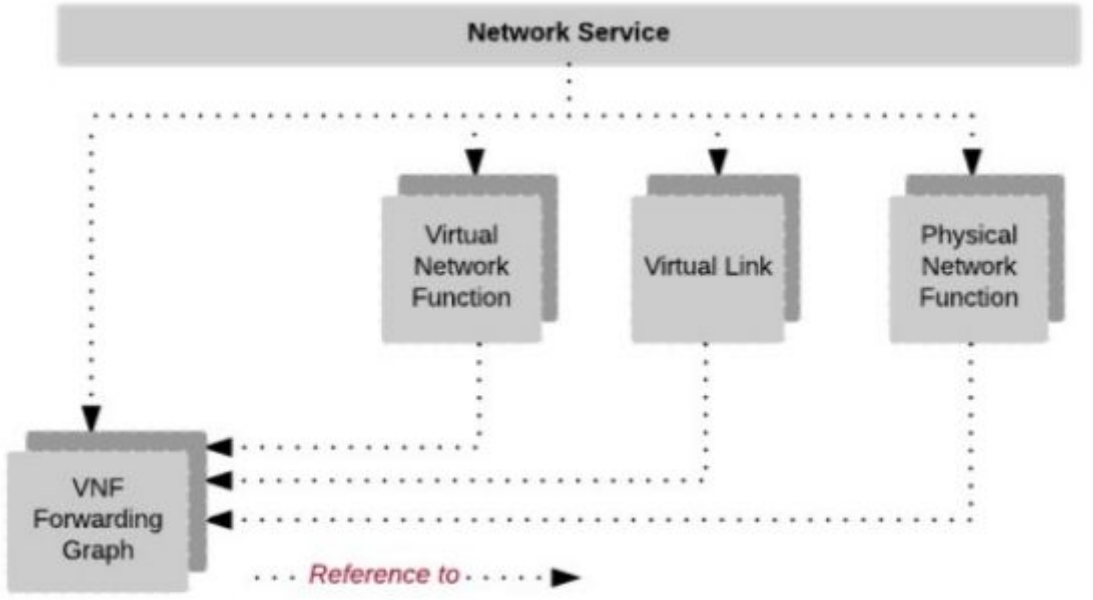


Figure 3.1: Structure of a MANO NS

The PNFD defines a physical network function, specifying the interconnections in the case the NS needs a physical device to expand the network. It will not be analyzed thoroughly because it is not yet supported by OSM platform and the focus is set on the virtual perspective of the topic.

### 3.1.1 - Network Service Descriptor (NSD)

The Network Service Descriptor (NSD) is the top-level structure that defines the topology of the network, wrapping all the references to the descriptors corresponding to the other components.

Figure 3.2 shows the high-level object model for the NSD. The task of this type of file is to permit the instantiation of a NS by referencing to the information contained in its lines.
In fact the NSD keeps the references to one or more VNFD connection points. The VNFs containing these virtual interfaces are connected with a single or multiple VLs, while the VNFFGD establishes the data flow around the network.

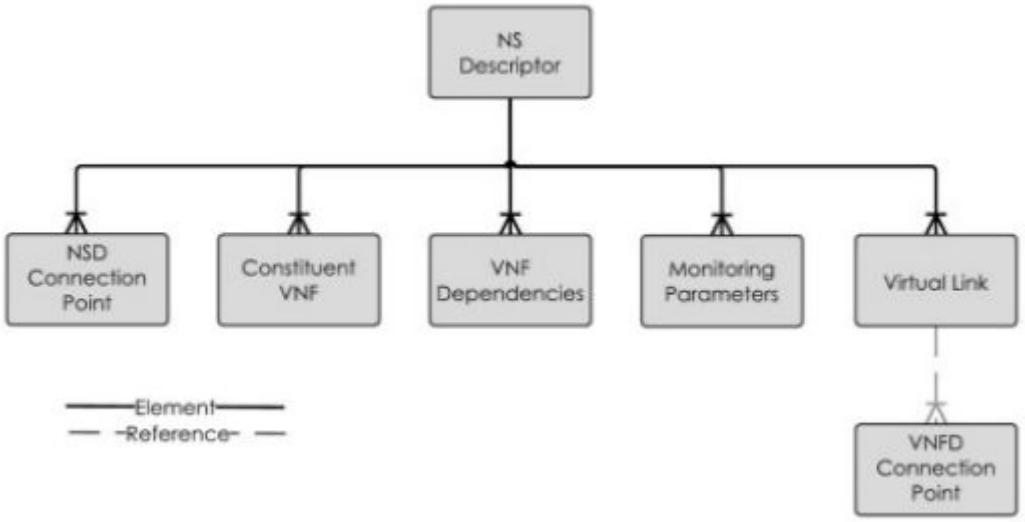


Figure 3.2: NSD object model.

## 3.1.2 - Virtual Network Function Descriptor (VNFD)

The Virtual Network Function Descriptor is a deployment template that defines the behaviour of a single VNF. Initially it is used by the VNF Manager to instantiate the lifecycle manager of the related function, later it provides information to the NFV Orchestrator in order to manage and orchestrate network services and virtualized resources on the NFV Infrastructure.
It is important to highlight the fact that a VNF is made up by a set of VNF Components (VNFC), that are pieces of software packaged together to make a more complex architecture. These elements are realized using a computing resource from the VIM, which could be either a virtual machine or a container. In order to run

those components a Virtual Deployment Unit (VDU) must be defined, indicating also the necessary information regarding storage, memory, CPU and networking resources.
In this file are reported:

- VNF images, containing the application and the launchpad
- Connection points and virtual links required to establish connection between VNFCs and between VNFs and the outside network
- VDU specifying compute, storage, memory and network resources
- Platform resource requirements like CPU, RAM, interfaces and network
- EPA characteristics and performance capabilities

To let the VNF to interface with other instances, it has available internal and external connection points, corresponding to the virtual interfaces used by the VM/container. The internal ones are useful to establish connectivity between two or more VNFCs, while the external ones can be used to keep the communication with the outside network. Either internal or external, both of them are connected using virtual links. Each VL has references to two or more connection points.
Figure 3.3 is the representation of the high-level object model of the VNFD. This contains lists of VDUs, internal connection points, internal virtual links, and external connection points. The internal connection points and internal virtual links define how the VMs inside the VNF will be connected. The external connection points are used by the NSD to chain VNFs. The VDUs define the individual VNF components and capture information about VM image, VM flavor, and EPA attributes.

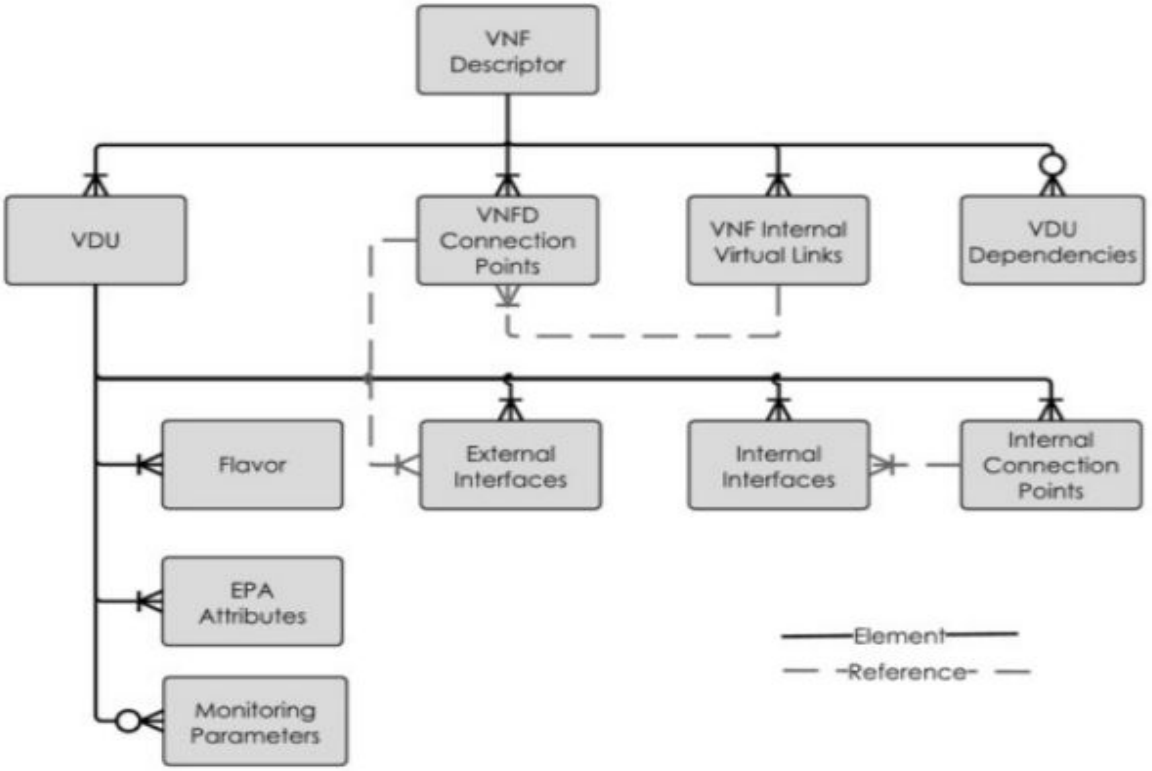


Figure 3.3: VNFD object model

### 3.1.3 - Virtual Link Descriptor (VLD)

A Virtual Link Descriptor (VLD) is a deployment template that describes the resource requirements needed for a link between VNFs, PNFs and endpoints of the network service. Even though it could seem obvious, it is important to highlight the fact that a NS cannot work without VLs.

The VLD provides a description of each Virtual Link. This type of information can be used by the NFVO to determine the appropriate placement of a VNF instance, and by the VIM responsible for managing the virtualized resources of the selected placement to determine the allocation of required virtualized resources on a host with adequate network infrastructure. The VIM can also use this information to establish the appropriate paths. The VLD describes the basic topology of the connectivity (E-LAN, E-Line, E-Tree) between one or more VNFs connected to this

VL and other required parameters (such as bandwidth and QoS class). The VLD connection parameters are expected to have similar attributes to those used on the ports on VNFs in ETSI standards. Therefore a set of VLs in a Network Service can be mapped to a Network Connectivity Topology (NCT). Figure 3.4 shows an example of a NCT described by the use of VLs referencing Connection Points of the VNFs and the NS.

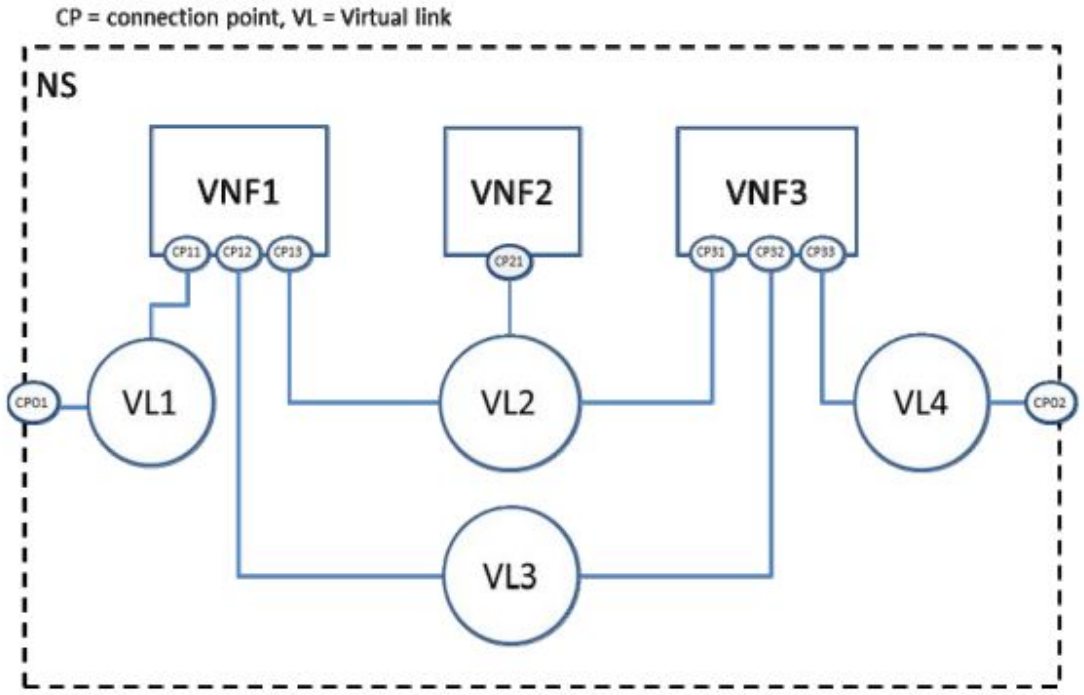


Figure 3.4: Example of a Network Connection Topology

## 3.1.4 - VNF Forwarding Graph Descriptor (VNFFGD)

A 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, PNFs and Virtual Links that connect these two. A VNFFG is a graph, specified by a network service provider, of bi-directional logical links that connect network function nodes, where at least one node is a VNF through which

network traffic is directed. A VNFFG model consists of a list of Rendered Service Path (RSP) and list of classifier components.
Figure 1.4 shows a graphical representation of three different VNFFG, distinguished by the three colors.
Up until now it is not possible to know how the VNFFG will track changes to the deployed instances, such as additional VNF instances being brought into service to handle scalability required for a traffic peak and migration of VNF workloads to an alternative infrastructure to enable operational maintenance or provide business continuity in the event of major link or site failure.

## 3.2 - OSM Descriptors

As briefly explained before, OSM uses yaml language for descriptor files due to its readability and efficiency.
The distinction between the main types of descriptors made before in this chapter is less evident in OSM platform, because only two files are required to instantiate a NS. In fact, to deploy this service it is necessary to onboard only the NSD and the VNFD wrapped into two different descriptor packages and then the NFVO will do the instantiations needed. The reason why only two descriptor files are required is because the VLDs and the VNFFGDs constituting the network topology are defined in the NSD template.

To give a brief introduction to the structure of descriptors, a NSD and a VNFD will be reported, showing how a basic NS made of a single VNF is instantiated just providing these two files.

The VNFD is:

```
1  vnfd-catalog:
2      vnfd:
3      -   id: basic-vnf
```

```
4          name: basic-vnf
5          short-name: basic-vnf
6          version: '1.0'
7          description: A VNF in a VDU
8          logo: osm.png
9          connection-point:
10         -   name: vnf-cp0
11             type: VPORT
12         vdu:
13         -   id: basic-vm
14             name: basic-vm
15             image: ubuntu1604
16             count: '1'
17             vm-flavor:
18                 vcpu-count: '1'
19                 memory-mb: '1024'
20                 storage-gb: '10'
21             interface:
22             -   name: vdu-eth0
23                 type: EXTERNAL
24                 virtual-interface:
25                     type: VIRTIO
26                 external-connection-point-ref: vnf-cp0
27         mgmt-interface:
28             cp: vnf-cp0
```

It could be unnecessary to say, but it is important to notice that the `id` must be unique in the catalog, otherwise it will be refused the time we try to onboard it onto the platform. It is fundamental to specify that the `image` field of the `vdu` must be given the same name belonging to the disk image uploaded on the VIM connected to the platform.

The NSD is:

```
1  nsd-catalog:
2      nsd:
3      -   id: basic-ns
4          name: basic-ns
5          short-name: basic-ns
6          description: A NS with a VNF and a VL
7          version: '1.0'
8          logo: osm.png
9          constituent-vnfd:
10         -   vnfd-id-ref: basic-vnf
11             member-vnf-index: '1'
12         vld:
13         -   id: mgmt-net
14             name: mgmt-net
15             short-name: mgmt-net
16             type: ELAN
17             mgmt-network: 'true'
18             vnfd-connection-point-ref:
19             -   vnfd-id-ref: basic-vnf
20                 member-vnf-index-ref: '1'
21                 vnfd-connection-point-ref: vnf-cp0
```

As the first lines are similar to the descriptor before, this type of file has also the reference to the VNF and the definition of the VLD, establishing connection between the management network of the service and the connection point of the VNF.

In the next chapter more complex files will be created and analyzed, providing the necessary information to create the desired topology the easiest way possible.

# 4. Complex Networks Implementation

In this chapter we will examine some examples of network topologies and try to create a NSD and one or more VNFD for each one, describing the most important lines of code and explaining how it works.
What might jump out in the next files is the fact that in none of them there is a reference to any VNFFGD: this is because that model is specified by network service providers and up until now there are no examples of how it works in OSM.

Thanks to the experience gained during the internship handling OSM platform, I could understand also how it works when there are different VMs and it is necessary to establish a connection between them to make the whole system work. Therefore it is important, as a first step, to create all the VIMs needed to store and run the functions required for the services. Once up and running, by using special commands from CLI, it is possible to set up connectivity between the orchestrator and the VIMs, referencing them by their IP address in the LAN.

It is essential to notice that actually it is not possible to connect VMs in different LANs, because of the fact that OSM does not support it and would not be able to keep trace of the path to the external VIMs, but the second example is thought as if it is possible to create a system working on more than one LANs interconnected.

For many reasons, there was not an infrastructure able to let me try if those elements are in fact working or not, but this is an attempt of creating something working by taking advantage of all the study and analysis made in the chapters before.

## 4.1 - Local Area Network

In this first example we will create a NS between two end points belonging to the same LAN with a VNF in the middle. In this setting OSM machine, VIM machine and the two end-points belong to the same network.

This is the VNF:

```
1. vnfd-catalog:
2.     vnfd:
3.     -   id: lan-vnf
4.         name: lan-vnf
5.         short-name: lan-vnf
6.         version: '1.0'
7.         description: A VNF in a VDU
8.         logo: osm.png
9.         connection-point:
10.          -   name: vnf-cp0
11.              type: VPORT
12.          vdu:
13.          -   id: lan-vm
14.              name: lan-vm
15.              image: vnf-vim1
16.              count: '1'
17.              vm-flavor:
18.                  vcpu-count: '1'
19.                  memory-mb: '1024'
20.                  storage-gb: '10'
21.              interface:
22.              -   name: vdu-eth0
23.                  type: EXTERNAL
```

```
24.                 virtual-interface:
25.                     type: VIRTIO
26.                 external-connection-point-ref: vnf-cp0
27.         mgmt-interface:
28.             cp: vnf-cp0
```

This VNFD is quite standard and really basic.
In line 23 we can see the type of the interface: in this example it is `EXTERNAL` because it allows to connect other VNFs or end-points for the chaining. In the case we see the type `INTERNAL`, that means that that one will be used to connect VNFCs to compone the VNF.
Lines 27 and 28 specify that the interface with which the VNF could be managed is the one corresponding to the connection point `vnf-cp0` previously created.

This is the NSD:

```
1. nsd-catalog:
2.     nsd:
3.     -   id: lan-ns
4.         name: lan-ns
5.         short-name: lan-ns
6.         description: A NS with a VNF and two end-points
7.         version: '1.0'
8.         logo: osm.png
9.         connection-point:
10.        -   id: end-p1
11.            name: end-p1
12.            short-name: end-p1
13.            type: VPORT
14.        -   id: end-p2
15.            name: end-p2
16.            short-name: end-p2
17.            type: VPORT
```

```
18.         constituent-vnfd:
19.         -   vnfd-id-ref: lan-vnf
20.             member-vnf-index: '1'
21.         vld:
22.         -   id: mgmt-net
23.             name: mgmt-net
24.             short-name: mgmt-net
25.             type: ELAN
26.             mgmt-network: 'true'
27.             vnfd-connection-point-ref:
28.             -   vnfd-id-ref: lan-vnf
29.                 member-vnf-index-ref: '1'
30.                 vnfd-connection-point-ref: vnf-cp0
```

Between lines 9 and 17 we can see the instantiation of two connection points, corresponding to those ports where the end-points can connect to use the service.
In lines 18, 19 and 20 is specified which vnf must be used to create the service desired.
From line 21 until the end a VL is instantiated to let the orchestrator manage the VNF and its life cycle.

## 4.2 - Local Area Networks Interconnected

In this second example we will create a NS between two LANs and try to create a service chain between two VNFs. It is important to notice again that this is still not possible, due to the limited capabilities of OSM.
Now OSM machine, the two VIM machines and the two end-points belong to different networks.

This is the NSD:

```
1. nsd-catalog:
2.     nsd:
3.     -   id: intlan-ns
4.         name: intlan-ns
5.         short-name: intlan-ns
6.         description:A NS with two VNFs and two end-points
7.         version: '1.0'
8.         logo: osm.png
9.         connection-point:
10.        -   id: end-p1
11.            name: end-p1
12.            short-name: end-p1
13.            type: VPORT
14.        -   id: end-p2
15.            name: end-p2
16.            short-name: end-p2
17.            type: VPORT
18.        constituent-vnfd:
19.        -   vnfd-id-ref: lan1-vnf
20.            member-vnf-index: '1'
21.        -   vnfd-id-ref: lan2-vnf
22.            member-vnf-index: '2'
23.        vld:
24.        -   id: mgmt-net
25.            name: mgmt-net
26.            short-name: mgmt-net
27.            type: ELAN
28.            mgmt-network: 'true'
29.            vnfd-connection-point-ref:
30.            -   vnfd-id-ref: lan1-vnf
31.                member-vnf-index-ref: '1'
```

```
32.                 vnfd-connection-point-ref: vnf-cp0
33.             vnfd-connection-point-ref:
34.             -   vnfd-id-ref: lan2-vnf
35.                 member-vnf-index-ref: '2'
36.                 vnfd-connection-point-ref: vnf-cp1
```

There are not big changes in the NSD, because primarily it has to deal with virtual resources and references to other descriptors. So beyond id and naming, there are differences in lines 21-22 and 33 to 36, with the addition of respectively a VNF and a reference to the connection point for the creation of the service chain through the VL.

While the first VNF might be identical to the one reported in the first example, this is the second VNF:

```
1. vnfd-catalog:
2.     vnfd:
3.     -   id: lan2-vnf
4.         name: lan2-vnf
5.         short-name: intlan2-vnf
6.         version: '1.0'
7.         description: A VNF in a VDU
8.         logo: osm.png
9.         connection-point:
10.          -   name: vnf-cp1
11.              type: VPORT
12.          vdu:
13.          -   id: lan-vm
14.              name: lan-vm
15.              image: vnf-vim2
16.              count: '1'
17.              vm-flavor:
```

```
18.                 vcpu-count: '1'
19.                 memory-mb: '1024'
20.                 storage-gb: '10'
21.             interface:
22.             -   name: vdu-eth0
23.                 type: EXTERNAL
24.                 virtual-interface:
25.                     type: VIRTIO
26.                 external-connection-point-ref: vnf-cp1
27.         mgmt-interface:
28.                 cp: vnf-cp1
```

The meaningful distinction of this descriptor from the previous one can be seen in line 15: that specifies the image name of the VIM running in one of the virtual machines and each one gets a different name when onboarded. This allows to understand where that specific VNF will be instantiated, leaving the programmer the decision.

# 5. Conclusions

The development of Management and Orchestration in Network Function Virtualization environments could be the solution for operators and service providers to ease the process of management and maintenance of their networks and it would also narrow down the amount of money they need to spend in infrastructures.

Being developed for only few years, the OSM platform still lacks of important features and its setup is very burdensome too. The lack of a well-structured documentation is also a big concern, because it does not allow new passionate developers to understand how to help the community with own working lines of code.

As explained in the chapter before, it is only possible to create basic network topologies and it is also needed an ad hoc customization of the testing environment. YAML descriptor files are really useful for their human readability and the possibility of using references is a great powerful aspect.

Obviously there is the problem of the seamlessly transition from physical networks to this innovation, but it is still at its first steps because of the fact that even now there are many challenges regarding their initial coexistence.

# Acknowledgement

There is so much pride and satisfaction in myself about achieving this first important goal in my life that I cannot even explain.
And obviously I need to thanks to all those people who were there by my side supporting and helping me go through many ups and downs.

First of all I just want to start saying a big thanks to Prof. Callegati, who made me develop a passion in computer networking and supervised me through the process of writing this thesis with willingness and accuracy.

The best moments spent at University were with my “Piscologi” (not spelt wrong), Modo, Pippis and Simo, who made me enjoy even the gloomiest day when the boiler in lecture hall stopped working in mid December (Max 10°C, no kidding). The continuous support and help got from you was unmatchable.

The people I owe a lot for how their presence has been felt since high school times are my “Regine di Cuori”, Loris, Vero, Marco and Cami, whose friendship never vacillated one second and they have been supportive every time I needed for a really long time (and I could bet they will carry on).

How can I forget to mention the gaming guys from “FTP”, Alfius, Baba, Fragolò, Teo and Matteo who are the reason I often laugh until tears while playing video games with them. Some would not believe this, but even though they are scattered all around Italy I love them as they are always physically by my side, because the friendship bond we have is for real.

And here comes the gang, the people I spend the most of the time with, those who make me feel like a kid when it is time to have fun but also like a lucky man when it is time to deal with real things. My “Autisti”, Agnesina, Ari, Rodilò, Pepu, Elenina, Iris, Giada, Lalli, Nico A., Nico B., Tia, Vale and Giulione, are those crazy people who always promised me to make a “scooterata alla villa” but never did (was the lack of fliers the problem? duh). Anyway thank you for the patience and backing you gave me when I needed, especially when it did not come to study.

Of course I have to thank all the lads at Promosport (#GoPromo) and all those I did not mention for the hurry in writing this part.

The best of my thanks goes to my relatives and especially I must give credit also to my family for this graduation.

I want to say thank you to my mother Gloriana and my father Stefano for the economic and motivational support they gave me from day one, always trusting in me and respecting my decisions even though they did not always agree with me. Despite of all the many obstacles and hard moments we have been through they always showed me love and I will always be thankful for the life lessons they gave me. And why not, thanks also for the patience and understanding in those moments when I go crazy at home and I start making stupid jokes and noise just to have fun.

And obviously, even though we are just like the cat and the mouse, I want to say thank you to my brother Filippo, who has always been a model for me for the dedication he put in its studies and everyday life. He still helps me taking my decisions with valuable advices and he shows his support to me day in and day out.

Finally, the conclusive thanks goes to “Mimmo”, the main reason I ended up being so fool (obviously in a positive way).

# Bibliography

**Online**

ETSI NFV, https://www.etsi.org/technologies-clusters/technologies/nfv, 24/08/2018

ETSI Architectural Framework, https://docbox.etsi.org/ISG/NFV/Open/Publications_pdf/Specs-Reports/NFV%20002v1.2.1%20-%20GS%20-%20NFV%20Architectural%20Framework.pdf, 01/09/2018

ETSI MANO, https://www.etsi.org/deliver/etsi_gs/NFV-MAN/001_099/001/01.01.01_60/gs_NFV-MAN001v010101p.pdf, 13/09/2018

ETSI Network Service Template Specification, https://docbox.etsi.org/ISG/NFV/Open/Publications_pdf/Specs-Reports/NFV-IFA%20014v3.1.1%20-%20GS%20-%20Network%20Service%20Templates%20Spec.pdf, 24/09/2018

Open Source Mano, https://osm.etsi.org/, 24/08/2018

OSM Information Model, https://osm.etsi.org/wikipub/images/2/26/OSM_R2_Information_Model.pdf, 24/09/2018

OSM Release Three - Technical Overview, https://osm.etsi.org/images/OSM-Whitepaper-TechContent-ReleaseTHREE-FINAL.PDF, 17/09/2018

OSM Release Four - Technical Overview, https://osm.etsi.org/images/OSM-Whitepaper-TechContent-ReleaseFOUR-FINAL.pdf, 17/09/2018

Reference VNF and NS Descriptors, https://osm.etsi.org/wikipub/index.php/Reference_VNF_and_NS_Descriptors, 26/09/2018

A Cheat Sheet for Understanding “NFV Architecture”, https://www.telcocloudbridge.com/blog/a-cheat-sheet-for-understanding-nfv-architecture/, 01/09/2018

Software-Defined Networking, https://www.opennetworking.org/sdn-definition/, 28/09/2018

What is Network Service Chaining?, https://www.sdxcentral.com/sdn/network-virtualization/definitions/what-is-network-service-chaining/, 10/09/2018

Software-Defined Networking, https://en.wikipedia.org/wiki/Software-defined_networking, 28/09/2018

Network Function Virtualization, https://en.wikipedia.org/wiki/Network_function_virtualization, 24/08/2018

YANG, https://en.wikipedia.org/wiki/YANG, 26/09/2018

YAML, https://en.wikipedia.org/wiki/YAML, 26/09/2018

OPNFV, https://www.opnfv.org/, 01/09/2018

# Acronyms

| | |
|---|---|
| API | Application Programming Interface |
| BSS | Business Support System |
| CLI | Command Line Interface |
| EM | Element Management |
| EMS | Element Management System |
| EPA | Enhanced Platform Awareness |
| ETSI | European Telecommunications Standards Institute |
| GUI | Graphical User Interface |
| IP | Internet Protocol |
| ISA | Instruction Set Architecture |
| JSON | JavaScript Object Notation |
| LAN | Local Area Network |
| MAC | Media Access Control |
| MANO | Management ANd Orchestration |
| NFV | Network Functions Virtualization |
| NFVI | Network Function Virtualization Infrastructure |
| NFVO | Network Functions Virtualization Orchestrator |
| NIC | Network Interface Card |
| NS | Network Service |
| NSD | Network Service Descriptor |
| ODL | Open DayLight |
| ONOS | Open Networking Operating System |
| OPNFV | Open Platform for Network Function Virtualization |
| OS | Operating System |
| OSM | Open Source Mano |
| OSS | Operations Support System |

| | |
|---|---|
| PCIe | Peripheral Component Interconnect Express |
| PNF | Physical Network Function |
| PNFD | Physical Network Function Descriptor |
| RBAC | Role Based Access Control |
| RO | Resource Orchestrator |
| SDN | Software Defined Networking |
| SO | Service Orchestrator |
| UI | User Interface |
| URI | Uniform Resource Identifier |
| VCA | Virtual Network Functions Configuration and Abstraction |
| VDU | Virtual Deployment Unit |
| VIM | Virtualized Infrastructure Manager |
| VL | Virtual Link |
| VLAN | Virtual Local Area Network |
| VLD | Virtual Link Descriptor |
| VM | Virtual Machine |
| VMM | Virtual Machine Monitor |
| VNF | Virtual Network Function |
| VNFC | Virtual Network Function Component |
| VNFD | Virtual Network Functions Descriptor |
| VNFFG | Virtual Network Functions Forwarding Graph |
| VNFFGD | Virtual Network Functions Forwarding Graph Descriptor |
| VNFM | Virtual Network Functions Manager |
| VPN | Virtual Private Network |
| YAML | YAML Ain’t Markup Language |