Small cEllS coordinAtion for Multi-tenancy and Edge services

Grant Agreement No.671596

Topic: H2020-2014-ICT-14
Advanced 5G Network Infrastructure for the Future Internet
Research and Innovation Action

Deliverable D5.1
Description of CESC abstraction model

Document Number:	H2020-5GPPP-GA No.671596/*WP5/D5.1/30.06.2016*
Contractual Date of Delivery:	*01.07.2016*
Editor:	Pietro Paglierani (Italtel SpA)
Work-package:	*WP5*
Distribution / Type:	Public (PU) / Report (R)
Version:	*1.0*
Total Number of Pages:	94
File:	*SESAME_Deliverable D5.1_v1.0_Final*

Abstract

This deliverable covers the activities carried out in Task 5.1 and related to the definition of the SESAME Cloud Enabled Small Cell (CESC) abstraction model. A state-of-the-art survey is first presented, aimed at analysing the most common models that could be used in SESAME.
In particular, the ETSI information model proposed in the context of Network Function Virtualisation is considered. The most relevant data modelling languages are then analysed: TOSCA, YANG, the FP7 T-Nova project models and OpenStack Heat Orchestration Templates are discussed, and some meaningful examples are reported. Finally, based on the comparison of the available solutions, the SESAME descriptors are outlined and discussed.

Version History

Version	Date	Comments, Changes, Status	Authors, contributors, reviewers
0.1	04/05/2016	Initial draft by Italtel	ITL
0.2	11/05/2016	Updates in section 4	ATOS, NCSRD and ORION
0.3	13/05/2016	Updates in section 2 and in ToC	ATOS
0.4	23/05/2016	Updates in sections 2.1, 2.2, 2.3, 2.4	FLE
0.5	26/05/2016	Added section 4.4	VOSYS
0.6	08/06/2016	Added section 3.1	OTE
0.7	13/06/2016	Updates in section 4.1, 4.2 and 4.3	FLE
0.8	20/06/2016	Reviewed version and added additional ITL contributions	ITL
0.9	21/06/2016	EHU contribution to 3.4.4 and section 4	EHU
0.10	22/06/2016	Updates in section 5	ATOS
0.11	24/06/2016	Updates in section 5.3 and 2.7	OTE
0.12	26/06.2016	Updates in abstract, conclusions, vnfd, pnfd and nsd schema	ITL
0.13	27/06/2016	General review	i2CAT
0.14	28/06/2016	Review	VOSYS
0.15	29/06/2016	Review	OTE
0.16	30/06/2016	Including Annex B	i2CAT
1.0	01/07/2016	Final Full Editoria and Conceptual Review by the Project Coordinator; Submission to the Commission	OTE

Contributors

First Name	Last Name	Partner	Email
Pietro	Paglierani	ITL	pietro.paglierani@italtel.com
Javier	García	ATOS	javier.garcial@atos.net
Ignacio	Labrador	ATOS	ignacio.labrador@atos.net
Michele	Paolino	VOSYS	m.paolino@virtualopensystems.com
Pavel	Bliznakov	VOSYS	p.bliznakov@virtualopensystems.com
Sebastien	Pinneterre	VOSYS	s.pinneterre@virtualopensystems.com
Charles	Turyagyenda	FLE	Charles.turyagyenda@uk.fujitsu.com
Begoña	Blanco	EHU	begona.blanco@ehu.eus
Jose Oscar	Fajardo	EHU	joseoscar.fajardo@ehu.eus
Evangelos	Sfakianakis	OTE	esfak@oteresearch.gr
Maria	Belesioti	OTE	mbelesioti@oteresearch.gr
Nikolas	Bompetsis	OTE	nbompetsis@oteresearch.gr
Ioannis	Chochliouros	OTE	ichochliouros@oteresearch.gr

Glossary

Acronym	Explanation
3GPP	Third Generation Partnership Project
ACK	Acknowledgement
AP	Application Protocol
API	Application Programming Interface
AWS	Amazon Web Services
CESC	Cloud Enabled Small Cell
CESCM	Cloud Enabled Small Cell Manager
CFN	CloudFormation
CoMP	Coordinated Multipoint
CP	Connection Point
CPU	Central Processing Unit
cps	call per second
CSAR	Cloud Service ARchive
DC	Data Centre
DL	Downlink
DPI	Deep Packet Inspection
EBNF	Extended Backus-Naur Form
EPC	Evolved Packet Core
ETSI	European Telecommunications Standards Institute
FG	Forwarding Graph
FP	Forwarding Path
FP	Framework Programme
FP7	7th Framework Programme
FPGA	Field programmable Gate Array
GA	Grant Agreement
GPRS	General Packet Radio Service
GPU	Graphics Processing Unit
GTP	GPRS Tunnelling Protocol
GW	Gateway
H2020	Horizon 2020
HeNB	Home eNodeB
HOT	Heat Orchestration Template
HTTP, http	Hypertext Transfer Protocol
HW, hw	Hardware
I/O, i/o	Input/Output
IaaS	Infrastructure as a Service
ICT	Information and Communication Technology
ID, id	Identifier
IETF	Internet Engineering Task Force
IP	Internet Protocol
IPsec, IPSEC	Internet Protocol Security
IT	Information Technology
JSON	JavaScript Object Notation
KPI	Key Performance Index
KVM	Kernel Virtual Machine
LAN	Local Area Network
Light DC	Light Data Center
LTE	Long Term Evolution
µS	micro server
MAC	Medium Access Control
MANO	MANagement and Orchestration

MIMO	Multiple Input, Multiple Output
NCP	Network Configuration Protocol
NCT	Network Connectivity Topology
NF	Network Function
NFP	Network Forwarding Path
NFV	Network Functions Virtualisation
NFV	Network Function Virtualization
NFVI	NFV Infrastructure
NFVO	NFV Orchestrator
NMS	Network Management System
NO	Network Oerator
NS	Network Service
NSD	Network Service Descriptor
OASIS	Organization for the Advancement of Structured Information Standards
OCCI	Open Cloud Computing Interface
OSM	Open Source MANO
PaaS	Platform as a Service
PDCP	Packet data Convergence Protocol
PHY	Physical Layer
PNF	Physical Network Function
PNFD	Physical Network Function Descriptor
PoC	Proof of Concept
PPP	Public-Private Partnership
QEMU	Quick EMUlator
QoS	Quality of Service
RAM	Random Access Memory
RAN	Radio Access Network
REA	Research Executive Agency
REST	Representational State Transfer
RF	Radio Frequency
RFC	Request for Comments
RIA	Research and Innovation Action
RLC	Radio Link Control
RRC	Radio Resource Control
RRM	Radio Resource Management
S1AP	S1 Application Protocol
SaaS	Software as a Service
SC	Small Cell
SCNO	Small Cell Network Oerator
SCTP	Stream Control Transmission Protocol
SDN	Software Defined Networking
SID	Shared Information/Data Model
SISO	Single Input, Single Output
SLA	Service Level Agreement
SMI	Structure of Management Information
SNMP	Simple Network Management Protocol
SOTA	State-of-the-Art
SSH, ssh	Secure Shell
SW	Software
TC	Technical Committee
TOSCA	Topology and Orchestration Specification for Cloud Applications
TU	Transcoding Unit
UDP	User Datagram Protocol
UE	User Equipment
UL	Uplink

vCPU	Virtual Central Processing Unit
vDPI	virtual Deep Packet Inspection
VDU	Virtual Deployment Unit
vEPC	Virtual EPC
VIM	Virtualized Infrastructure Manager
VL	Virtual Link
VLAN	Virtual Local Area Network
VLD	Virtual Link Descriptor
VM	Virtual Machine
VNF	Virtual Network Function
VNFC	Virtual Network Function Component
VNFD	Virtual Network Function Descriptor
VNFFG	VNF Forwarding Graph
VNFFP	VNF Forwarding Path
VNFM	Virtual Network Function Manager
VSCNO	Virtual Small Cell Network Operator
vTU	virtual Transcoding Unit
VXLAN	Virtual Extensible LAN
YAML	YAML Ain't Markup Language
YANG	Yet Another Next Generation
WP	Work Package
X2AP	X2 Application Protocol
XML	Extensible Markup Language
XSLT	Extensible Stylesheet Language Transformations

Table of Contents

List of Figures

List of Tables

1 Introduction

One of the main objectives of SESAME is the realization of a virtualised execution platform, the Light DC (Data Centre), shared by a cluster of Cloud Enabled Small Cells, so as to run Virtual Network Functions directly in the access infrastructure. To this end, Cloud Computing concepts must be necessarily included both in the CESCs and in their manager, the CESCM. In particular, there is the need of suitably modelling all the virtualised and physical resources made available by the CESCs, so as to assign them in a dynamic and effective way to the VNFs. On their turn, the VNFs must specify their needs in terms of resources, as well as their deployment options, security constraints, etc., in a specific document known as the “VNF Descriptor".

The formal description of virtual appliances and services in a virtualised infrastructure is a well-established topic, and many results are currently available. The ETSI MANO [1] provides a complete set of information models that can be used in Network Function Virtualisation frameworks. Modelling languages such as TOSCA [2] and YANG [3] are now widely used in real-life virtualised infrastructures, and various enhanced descriptors have been developed within research and open source initiatives, such as the FP7 T-Nova project [4]. Also, ad hoc tools that allow the translation of descriptors from one specific format to another are available [5].

However, while the adoption of such Cloud Computing concepts and tools is now well-established in any standard virtualised infrastructure, their applicability to the novel CESC architecture proposed by SESAME has been soon identified as one of the “open points” of the project. For this reason, the main objective of Task 5.1 has been the analysis of the portability of already available abstraction models to the SESAME CESC and CESCM. The problem of identifying eventual limitations imposed by the SESAME architecture, and in this case defining the needed extensions and modifications, has also been addressed.

This document summarises the activities carried out within Task 5.1, and reports the achieved results. The analysis has shown that, with the given SESAME CESC architecture defined in [6], the most common descriptors available in the literature can be used in SESAME, without specific modifications or proprietary extensions due to the integration of the Light DC with the Small Cell environment. In particular, descriptors in any of the analysed formats can be used in SESAME, provided that a translator to the internal SESAME descriptor format is available. The final choice of the SESAME internal descriptor format is an implementation issue, out of the scope of this document. Though, in the following parts, the most common models and languages are analysed, and examples that could be adopted by SESAME are reported and discussed. Finally, guidelines for the use of descriptors in SESAME are provided, which clarify “how the concepts discussed in this document should be used to introduce some fundamental Cloud concepts in the CESC architecture proposed by SESAME”.

1.1 Deliverable outline

The structure of the document is the following:

Section 1 offers a brief introductory overview, to clarify the objectives and the scope of the present document.

Section 2 introduces the major NFV concepts, and summarises the descriptors defined by the ETSI NFV Information Model.

Section 3 discusses the most common modelling approaches, i.e. the Heat Orchestration Template, and the TOSCA and YANG languages; moreover, the extensions introduced by the FP7 T-NOVA project to the ETSI data model are presented.

In *Section 4*, the different types of Network Functions managed in SESAME are considered, i.e. Service VNF, Small Cell VNF and PNF.

Section 5 specifies the possible solutions that can be adopted in SESAME, summarising pros and cons for each.

Finally, *Section 6* summarises the key issues discussed in the document and offers conclusions.

2 The ETSI NFV Information Model

In the ETSI MANO specification [1], to describe a Network Service and the components comprising the Network Service, information elements representing those components are introduced. There are five information elements defined:

- The top level Network Service (NS) information element;
- VNF Forwarding Graph (VNFFG) information element;
- Virtual Link (VL) information element;
- Virtualised Network Function (VNF) information element;
- Physical Network Function (PNF) information element.

One of the ways the information elements can be used is as descriptors in a catalogue or template context. In the following sub-sections we provide a brief description of those information elements descriptors.

2.1 Network Service Descriptor

A Network Service Descriptor (NSD) is a deployment template for a Network Service referencing all other descriptors which describe components that are part of that Network Service. It is used by the NFV Orchestrator to instantiate a Network Service, which would be formed by one or more VNF Forwarding Graphs, VNFs, PNFs and VLs. The NSD also describes deployment flavours of Network Service.

The following are the most relevant information elements in the NSD:

- List of *VNFs* which are part of the Network Service.
- List of *VNF forwarding graphs* which are part of the Network Service.
- List of *Virtual Links* which are part of the Network Service.
- List of the Network Service functional scripts/workflows for specific *lifecycle events* (e.g. initialization, termination, scaling).
- List of *dependencies* between VNFs.They are defined in terms of source and target VNFs (i.e. target VNF "depends on" source VNF).
- List of *monitoring parameters* which can be tracked for a Network Service. These can be network service metrics that are tracked for the purpose of meeting the network service availability contributing to SLAs (e.g. NS downtime). Also, they can be used for specifying different deployment flavours for the Network Service in Network Service Descriptor, and/or to indicate different levels of network service availability. Examples include specific parameters such as calls-per-second (cps), number-of-subscribers, no-of-rules, flows-per-second, etc.
- List of service KPI parameters and its requirement for each *deployment flavour* of the Network Service being described (for example, there could be a flavour describing the requirements to support a vEPC with 300k cps and another flavour describing the requirements to support a vEPC with 500k cps).
- List of *connection points* which act as endpoints of the Network Service. This can, for example, be referenced by other elements as an endpoint.
- The *auto-scale policy*. This represents the policy meta-data, which may include the criteria parameter & action-type. The criteria parameter should be a supported assurance parameter. An example of such a descriptor could be:
    - Criteria parameter: calls-per-second.
    - Action-type: scale-out to a different flavour ID.
- List of *PNFs* which are part of the Network Service.
- *Identity* of the Network Service Descriptor.
- Provider or *vendor* of the Network Service.

- *Version* of the Network Service Descriptor.
- *Security signature* of the Network Service Descriptor. The particular hash algorithm used to compute the signature, together with the corresponding cryptographic certificate to validate the signature.

2.2 VNF Forwarding Graph

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 and PNFs and Virtual Links that connect them.

The following Figure 1 shows an example of two VNFFGs established on top of a network topology. As part of the VNFFG, a Network Forwarding Path (NFP) is an ordered list of connection points that determines a chain of VNFs from the entry point to the exit. VNFFG1 has two NFPs (VNFFG1:NFP1 and VNFFG1:NFP2) whereas VNFFG2 only has a single NFP (VNFFG2:NFP1):

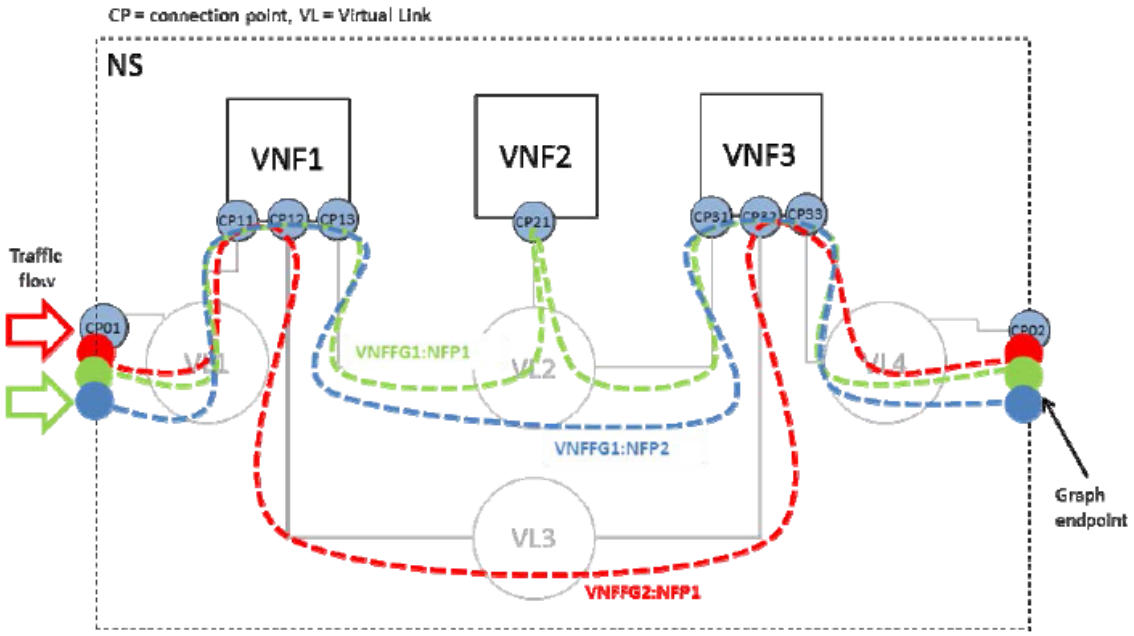

Figure 1: Example of VNFFGs

VNF Forwarding Graphs (VNFFGs) are expected to be developed by service providers or their systems integration partners, and could be customized from industry specified templates that encapsulate the common telecommunication service patterns.

The VNF Forwarding Graph Descriptor contains the following main information elements:

- Count of the *Virtual Links* (dependent_virtual_link elements) used by the VNFFG.
- Count of the *external endpoints* (connection_point elements) included in this VNFFG.
- List of *Network Forwarding Paths* (this is tuple containing a reference to a Connection Point in the NFP and the position in the path).
- List of *Connection Points* forming the VNFFG, including Connection Points attached to PNFs.
- Reference to the constituent *VNFs,* i.e. VNFD.
- *Identity* of the VNFFG.
- *Vendor* generating the VNFFG.
- *Name, version and description* of the service this VNFFG is describing.

- *Security signature* of the VNFFG. The particular hash algorithm used to compute the signature, together with the corresponding cryptographic certificate to validate the signature.

Likewise, the following information elements are included in the Network Forwarding Graph Descriptor:

- The policy or rule to apply to the traffic of the NFP.
- List of Connection Points that define the path.

2.3 VLink descriptor

A Virtual Link Descriptor (VLD) is a deployment template which describes the resource requirements that are needed for a link between VNFs, PNFs and endpoints of the Network Service, which could be met by various link options that are available in the NFVI.

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 virtualised resources of the selected placement to determine the allocation of required virtualised resources on a host with adequate network infrastructure. The VIM, or another Network Controller, can also use this information to establish the appropriate paths and VLANs.

The VLD describes the basic topology of the connectivity[1] (e.g. E-LAN, E-Line, E-Tree) between one or more VNFs connected to this VL and other required parameters (e.g. bandwidth and QoS class).

The main attributes in the VLD are the following:

- The number of *endpoints* available on this Virtual Link.
- *Throughput of the link* (e.g. bandwidth of E-Line, root bandwidth of E-Tree, and aggregate capacity of E-LAN).
- List of *throughput of leaf connections* to the link (for E-Tree and E-LAN branches).
- List of *QoS* options available on the VL, e.g. latency, jitter, etc.
- *Test access facilities* available on the VL (e.g. none, passive monitoring, or active loopbacks at endpoints).
- A reference to an attached *Connection Point.*
- *Connectivity type* (e.g. E-Line, E-LAN, or E-Tree).
- *Identity* of the VLD.
- *Vendor* generating this VLD.
- *Version* of this VLD.
- *Security signature* of the VLD. The particular hash algorithm used to compute the signature, together with the corresponding cryptographic certificate to validate the signature.

2.4 VNF descriptor

A VNF Descriptor (VNFD) is a deployment template which describes a VNF in terms of its deployment and operational behaviour requirements. It is primarily used by the VNFM in the process of VNF instantiation and lifecycle management of a VNF instance. The information provided in the VNFD is also used by the NFVO to manage and orchestrate Network Services and virtualised resources on the NFVI.

[1] More information about various different connectivity schemes can be provided, *inter-alia*, in: http://www.mef.net/PDF_Documents/technical-specifications/MEF6-1.pdf

The VNFD also contains connectivity, interface and KPIs requirements that can be used by NFV-MANO functional blocks to establish appropriate Virtual Links within the NFVI between its VNFC instances, or between a VNF instance and the endpoint interface to the other Network Functions.

The VNFD contains the following main information elements:

- Set of elements related to a particular *VDU* (Virtual Deployment Unit)
- *Virtual Link*. Represents the type of network connectivity mandated by the VNF vendor between two or more Connection Points.
- *Connection Point (CP)*. Describes an external interface exposed by this VNF enabling connection with a Virtual Link.
- Set of functional scripts/workflows for specific *lifecycle events* (e.g. initialization, termination, graceful shutdown, scaling out/in, update/upgrade, VNF state management related actions to support service continuity).
- List of *dependencies* between VDUs defined in terms of source and target VDU, i.e. target VDU "depends on" source VDU. In other words, sources VDU shall exist before target VDU can be initiated/deployed.
- *Monitoring parameters*, which can be tracked for this VNF (such as calls-per-second, number-of-subscribers, no-of-rules, flows-per-second, VNF downtime, etc.).
- *Deployment Flavour.* Represents the assurance parameter(s) and its requirements for each deployment flavor of the VNF being described.
- *Auto-Scale Policy*. Represents the policy meta-data, which may include the criteria parameter and action-type. Example of such a descriptor could be:
    - Criteria parameter: calls-per-second.
    - Action-type: scale-out to a different flavour ID, if exists.
- *Identity* of this VNFD.
- *Vendor* generating this VNFD.
- *Version* of VNFD.
- *Version of VNF software* described by the descriptor under consideration.
- Reference to a *manifest file* that lists all files in the package.

2.5 Physical Network Function Descriptor

A Physical Network Function Descriptor (PNFD) 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.

Although the ETSI MANO specification defines how VNFD are on-boarded in a VNF Catalogue (as part of a VNF Package) it does not specify how or where PNFD are on-boarded; in that specification it is assumed that is out of scope since that information is not specific to NFV (see section 6.1 in [1]).

The PNFD main information element is the *list of connection points*; this element describes an external interface exposed by this PNF enabling connection with a Virtual Link.

2.6 VNF lifecycle model

The ETSI MANO specification defines two specific VNF lifecycle control interfaces:

a) The ***Lifecycle Management Interface***, which allows authorised consumers to perform lifecycle operations on VNFs (i.e. all operations needed to request and manage associations of NFVI Resources to a VNF, and maintain such associations in conformance

with the VNF Descriptor and authorised run-time changes, throughout the lifecycle of the VNF).

b) The ***Lifecycle Change Notification Interface***. This interface is used to provide runtime notifications related to the state of the VNF instance, as a result of changes made to VNF instance including (not limited to) changes in the number of VDUs, changing VNF configuration and/or topology due to auto-scaling/update/upgrade/termination, switching to/from standby, etc.

The operations which are accessible through the *Lifecycle Management Interface* are:

- **Instantiate VNF**. This operation allows creating a VNF instance.
- **Query VNF**. This operation allows retrieving VNF instance state and attributes. Attributes returned may include for example number and location of VMs allocated to the VNF instance.
- **Scale VNF**. This operation allows scaling (out/in, up/down) a VNF instance.
- **Check VNF instantiation** feasibility. This operation allows verifying if the VNF instantiation is possible.
- **Heal VNF.** This operation is used to request appropriate correction actions in reaction to a failure. This assumes operational behaviour for healing actions by VNFM has been described in the VNFD. An example may be switching between active and standby mode.
- **Update VNF software**. This operation allows applying a minor/limited software update (e.g. patch) to a VNF instance.
- **Modify VNF**. This operation allows making structural changes (e.g. configuration, topology, behaviour, redundancy model) to a VNF instance.
- **Upgrade VNF software**. This operation allows deploying a new software release to a VNF instance
- **Terminate VNF**. This operation allows terminating gracefully or forcefully a previously created VNF instance.

On the other hand, the *Lifecycle Change Notification Interface* just provides one operation: the Notify operation itself. This operation allows providing notifications on state changes of a VNF instance, related to the VNF Lifecycle. The details of the mechanism for registering for notifications (e.g. subscribe) and all other possible operations related to the actual notification delivery mechanism are deliberately left out of scope in the ETSI MANO specification. Also, guaranteeing delivery of notifications is considered as an implementation issue and deliberately left out of scope.

In practice, different management flows related to VNF lifecycle can be implemented by means of the interfaces described above. In the Annex B of the ETSI MANO specification [1] you can find a collection with different flows related to VNF Lifecycle management.

2.7 Management of Computing Resources

Regarding the virtualisation part of the SESAME project, it needs to manage the computing resources to provide and cover the principles for essential features of the virtualised network edge as well. To attain the decoupling of various network services from the backhaul portion that is using network virtualisation like SDN and NFV, SESAME provides these services through the CESC cluster and more specifically with the Light DC. According to SESAME architecture the CESC Cluster, Light DC and VIM will be part of the computing resources. In order to achieve the

network edge virtualisation the orchestration of the compute nodes in the CESC cluster, we will use OpenStack and the appropriate modules[2], such as Nova, Neutron, Heat etc.

The OpenStack project is an open source project Cloud Computing platform that supports all type of cloud environments. Also OpenStack provides a *de-facto* production environment solution for Infrastructure-as-a-Service (IaaS) through a variety of complemental services and modules. Each service and module offers an application programming interface (API).

[2] OpenStack is a free and open-source software platform for cloud computing, mostly deployed as an infrastructure-as-a-service (IaaS). More information about OpenStack components can be found, *for example*, at; https://www.openstack.org/software/

3 Data Modelling Languages

Orchestration is the ability to automate the deployment and configuration of infrastructure. More than just starting up virtual services, it also manages scripts used to add VMs to networks, stands up multiple servers together as a "stack," and even installs application software.

The most widely adopted way to effectively manage orchestration is the solution originally adopted by the Amazon Web Services (AWS): to develop declarative *templates* residing in simple and easily managed text files[3].

Beyond AWS, other different modelling languages are commonly proposed to manage orchestration, although the underlying concept is always the same: to manage the deployment and configuration of the cloud infrastructure in an automated way. So, one of the areas of interest regarding SDN and NFV technologies, is about data modelling and templating technologies.

3.1 OpenStack Heat Orchestration Template

3.1.1 Introduction

The OpenStack Foundation[4] has defined a standard, for specifying resources and the orchestrations for managing infrastructure and application lifecycles, called HEAT[5]. So, HEAT is an orchestration engine, included in OpenStack, for the provisioning and management of arbitrarily complex Infrastructure as a Service (IaaS) infrastructures.

HEAT implements an orchestration engine to "launch" multiple composite cloud applications based on the so called *Orchestration Templates* which basically are text files that are readable by humans and can be treated like code.

At provisioning time, the Heat engine interprets the template and deploys the corresponding infrastructure by orchestrating calls to the APIs of other OpenStack modules, as shown in Figure 2:

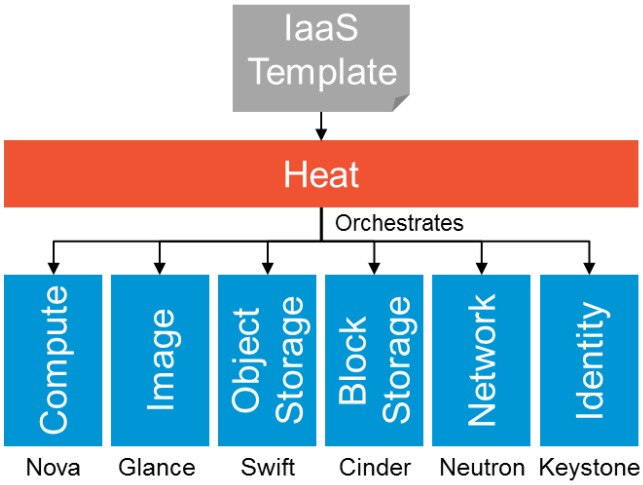

Figure 2: OpenStack modules

[3] On AWS they are called *CloudFormation* templates

[4] The OpenStack Foundation promotes the global development, distribution and adoption of the OpenStack cloud operating system. As the global independent home for OpenStack, the Foundation serves more than 30,000 Individual Members from over 170 countries around the world. More details are given in: https://www.openstack.org/foundation/

[5] https://wiki.openstack.org/wiki/Heat

An instance of a template is called a "stack". Heat templates can be modelled in two different formats:

- **CFN** (which stands for "CloudFormation"). This is the format used in AWS. Heat natively understands this format, simplifying application portability between AWS and OpenStack. CFN templates are usually (but not always) written in JSON language.

- **HOT** (Heat Orchestration Template). HOT templates are often (but not always) written in YAML format. YAML stands for "Yaml Ain't Markup Language," and is easy to read and understand by non-programmers. Its simplicity makes HOT templates accessible to system admins, architects, and other non-coders. Unlike CFN, HOT is not backward compatible with AWS.

In this document we will focus on the new HOT format, since it is considered the *de-facto* standard and it is considered to be the replacement of the legacy CFN format.

Independently from its format, a generic template is structured in three different sections, as shown in Figure 3:

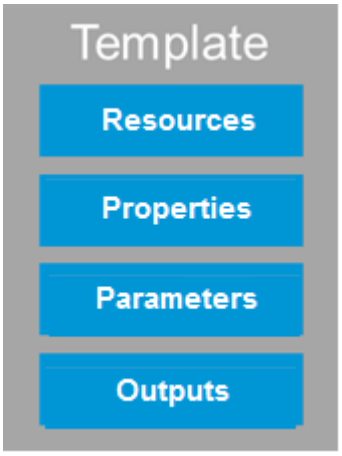

Figure 3: A HEAT template structure

- **Resources**: These are the details of your specific *stack*[6]. These are the objects that will be created or modified when the template runs. Resources could be Instances, Volumes, Security Groups, Floating IPs, or any number of objects in OpenStack. This is a mandatory field.
- **Properties**: These are specifics of your template. For example, you might want to specify your CentOS instance[7] in m1.small flavour[8]. Properties may be hard coded in the template, or may be prompted as Parameters.
- **Parameters**: Declaration of input parameters that have to be provided when instantiating the template (optional). In the HOT format, they appear before the Resources section and are mapped to Properties.

---

[6] As a whole, the "stack" is a collection of VMs and their associated configuration. But in Heat, the "Stack" term comes with a more specific meaning: It refers to a collection of resources. Resources could be instances (VMs), networks, security groups, and even auto-scaling rules.

[7] CentOS is a Linux distribution that is compatible with OpenStack. More information can be found at: https://wiki.centos.org/FrontPage

[8] For more details see, *for example*: http://docs.openstack.org/ops-guide/index.html

- **Outputs**: declaration of output parameters (e.g., IP addresses of the VMs) passed to the user once the template is instantiated (optional). It may be displayed in the dashboard, or revealed in certain command line commands.

The infrastructure resources that can be described using HEAT templates include: instances, floating IP addresses, volumes, security groups, users, etc. They can also specify relationships between resources (e.g. this volume is connected to that server) as well as some more advanced functionality such as instance high availability, instance auto-scaling and nested stacks. As a whole, they can be used to create the required cloud infrastructure in the correct order to completely launch the applications, and later, to manage the whole application lifecycle.

From an infrastructure point of view a generic VNF may be very complex, since it can be composed by several VNFCs, each of them implemented as a VM, connected through many different networks (both internal and external), not to mention additional storage/network resources (e.g., block storage volumes, load balancers, firewalls, etc.) and deployment policies.

Therefore, if not properly governed, both first-time deployment and lifecycle management of a VNF can be cumbersome and subject to error. In order to overcome these problems:

1) The infrastructure characteristics of a VNF, e.g., its topology, required resources and deployment policies, are described by means of HEAT templates;
2) The required infrastructure for a given VNF instance is requested by the VNFM to the VIM by means of OpenStack HEAT REST APIs. VIM credentials will be made available to the VNFM by the Orchestrator.

3.1.1.1 Resource Allocation in an ETSI MANO framework

For each data centre a Virtual Infrastructure Manager (VIM) guarantees the management and the allocation of the necessary virtual resources for the deployment of Virtual Network Functions (VNFs). These VNFs are composed of one or more Virtual Machines, which require the allocation of vCPUs, RAM and storage. The VIM is also responsible for creating and allocating the necessary network resources.

VNF Provisioning is the Orchestrator service responsible for interacting with the VIM in order to request a deployment of a VNF, and HEAT is the OpenStack orchestration engine [7] responsible for accepting requests and instantiating all the necessary infrastructure resources. HEAT provides a REST API [8] that the Orchestrator can use to request the creation of a VNF, the scaling of a VNF, information about a VNF and the termination of a VNF. Because HEAT only supports a specific kind of request in the form of a Heat Orchestration Template (HOT) to describe the required infrastructure, the Orchestrator must use the HOT Generator service to translate a VNF Descriptor (VNFD) to a HOT document. When HEAT receives this document it interacts with other OpenStack components to create the entire infrastructure in the correct order to launch the VNF. When infrastructure scaling is needed, the Orchestrator only needs to send the new HOT version to HEAT so that it updates the existing stack. For the termination of the VNF, the VNF Provisioning service sends a request to HEAT, which deletes all of the resources that have been allocated to the VNF.

The usage of the HEAT orchestration engine provides an advantage because it abstracts the interaction with all the other components of OpenStack however it does not have the concept of a NS.

3.1.2 HOT Templates Overview

HOT templates are commonly defined in YAML (*YAML Ain't Markup Language*) and follows the structure outlined below[9]:

```
heat_template_version: 2015-04-30
  # This key with value 2013-05-23 (or a later date) indicates that the YAML
```

[9] See [6], for additional details.

```
  # document is a HOT template of the specified version. This parameter tells Heat
  # not only the format of the template but also features that will be validated
  # and supported.

description:
  # This is an optional key for giving a description of the template

parameter_groups:
  # A declaration of input parameter groups and order. This section allows
  # for specifying how the input parameters should be grouped and the order to
  # provide the parameters in. This section is optional and can be omitted
  # when necessary.

parameters:
  # Declaration of input parameters. This section allows to specify input
  # parameters that have to be provided when instantiating the template.
  # The section is optional and can be omitted when no input is required.

resources:
  # Declaration of template resources. This section contains the declaration of
  # the single resources of the template. This section with at least one
  # resource should be defined in any HOT template, or the template would not really
  # do anything when being instantiated.

outputs:
  # Declaration of output parameters. This section allows for specifying
  # output parameters available to users once the template has been instantiated.
  # This section is optional and can be omitted when no output values are required.
```

The following shows a simple Heat template example where (besides the mandatory template version and the description fields) a stack with two VM instances is deployed. As you see, some properties can be explicitly specified (flavour), while others are defined using input parameters that should be provided by the user upon deployment:

```
heat_template_version: 2013-05-23

description: Simple template to deploy a stack with two virtual machine instances

parameters:
  image_name_1:
    type: string
    label: Image Name
    description: SCOIMAGE Specify an image name for instance1
    default: cirros-0.3.1-x86_64
  image_name_2:
    type: string
    label: Image Name
    description: SCOIMAGE Specify an image name for instance2
    default: cirros-0.3.1-x86_64
  network_id:
    type: string
    label: Network ID
    description: SCONETWORK Network to be used for the compute instance

resources:
  my_instance1:
```

```
    type: OS::Nova::Server
    properties:
      image: { get_param: image_name_1 }
      flavour: m1.small
      networks:
        - network : { get_param : network_id }
  my_instance2:
    type: OS::Nova::Server
    properties:
      image: { get_param: image_name_2 }
      flavour: m1.tiny
      networks:
        - network : { get_param : network_id }
```

Additional sample templates which demonstrate the core Heat functionality, together with related image-building templates and template-related scripts and conversion tools can be found in [9].

When it comes to running a Heat template there are different options:

- To save it as a file (using the 'yaml' extension such as *SimpleStack.yaml*). This way you can call the Heat engine from the command line tools or even REST calls.
- Using the OpenStack Horizon dashboard. This is the preferred method, especially for larger templates (for small templates one can just paste the code directly into Horizon).

3.1.3 OpenStack Workflow

OpenStack consists of several independent parts, the most important for the Deliverable 5.1 are described below. All services authenticate through a common Identity service. Most individual OpenStack's services interact with each other through public or private APIs. According to the deliverable, we illustrate an overview of the CESC Cluster and the integration with the OpenStack.

As shown in the following figure, SESAME CESC Cluster consists of two basic elements: (i) small cell, and; (ii) micro server. A detailed analysis of the Light DC architecture is given in [10]. Small cell constitutes the physical level of the SESAME approach. Micro server constitutes an important part of virtualisation in order to provide the essential features of the virtualised network edge.

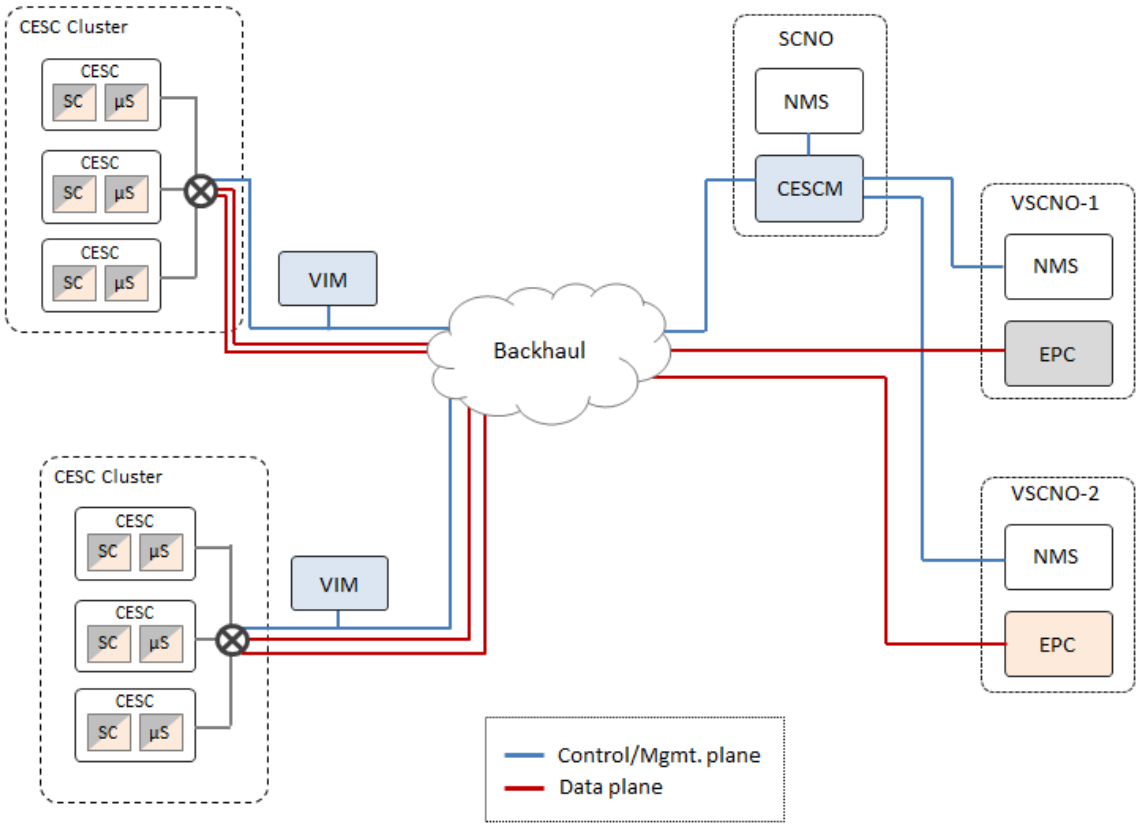

Figure 4: CESC Cluster – VIM – Backhaul Signalling Diagram

In SESAME architecture, micro server (µS) as a compute node must support virtualisation in order to provide the essential features of the SESAME network services. Any micro server (compute node) receives requests from the controller node (in our architecture VIM component) and instantiates and hosts virtual machine instances, according to specific templates, which are adjusted according to the specific VNF service requirements. Internally, (Nova) interacts with the Image service (Glance) to also select the appropriate disk image according to the received template.

The Compute service relies on a hypervisor to run virtual machine instances. OpenStack can use various hypervisors, but the SESAME approach will use QEMU[10]/KVM[11]. From the perspective of the Compute service functionality, the QEMU hypervisor is very similar to the KVM hypervisor. Both are controlled through *libvirt*[12], both support the same feature set, and all virtual machine images that are compatible with KVM are also compatible with QEMU. OpenStack Networking service (Neutron) provides an API that allows administrators to set up and define network connectivity and addressing from the VIM between the micro server and the backhaul. OpenStack Networking handles the creation and management of a virtual networking infrastructure including networks, switches, subnets, and routers for devices managed by the OpenStack Compute service (Nova). Advanced services such as firewalls or Virtual Private Networks (VPNs) can also be used. OpenStack Networking consists of the neutron-server, a database for persistent storage, and any number of plug-in agents, which provide other services

[10] QEMU is a generic and open source machine emulator and virtualizer. More information can be found at: http://wiki.qemu.org/Main_Page

[11] KVM is a full virtualization solution for Linux on x86 hardware containing virtualization extensions (Intel VT or AMD-V). More information can be found at: http://www.linux-kvm.org/page/Main_Page

[12] Libvirt is an open source API, daemon and management tool for managing platform virtualization. More information is given at: http://libvirt.org/

such as interfacing with native Linux networking mechanisms, external devices, or other SDN controllers such as the SESAME. OpenStack Networking interacts with Nova compute service to setup and configure the basic network parameters via VIM and its nova scheduler. At this point we are referring to the Control plane of signalling. After the network setup, the micro servers can transport the data plane through the vSwitch from/to the backhaul and the EPC.

3.1.4 Related Projects

3.1.4.1 Senlin

The mission of the Senlin project [11] is to provide a generic clustering service for an OpenStack cloud. Such a service is capable of managing the homogeneous objects exposed by other OpenStack components, such as Heat.

The primary features of the Senlin service are:

- A generic clustering/collection service for managing groups of homogeneous cloud objects on OpenStack. A user can create clusters of node and associate policy to such a cluster.
- The software interacts with other components of OpenStack so that clusters of resources exposed by those components can be created and operated.
- The software complements Heat project each other so Senlin can create and manage clusters of Heat stacks while Heat can invoke Senlin APIs to orchestrate collections of homogeneous resources.
- Based on an open design for action execution that can be extended to accommodate complex application deployment.
- Provides policies as plugins that can be used to specify how clusters operate. Example policies include creation policy, placement policy, deletion policy, load-balancing policy, scaling policy, etc.
- Can interact with all other OpenStack components via profile plugins. Each profile type implementation enables Senlin to create resources provided by a corresponding OpenStack service.
- Provides a set of APIs for managing cluster membership, e.g. add/remove nodes.
- Offers an asynchronous execution engine for ensuring the state consistency of clusters and nodes.

3.2 TOSCA

3.2.1 Introduction

TOSCA[13] (Topology and Orchestration Specification for Cloud Applications) is an important open cloud standard supported by a large and growing number of international industry leaders [2]. The first TOSCA version was released as an OASIS[14] standard in January 2014. OASIS is an international non-profit open standard consortium founded back in 1993 and devoted to the development, convergence and adoption of open standards for the global information society. OASIS members widely represent the marketplace of private and public sector technology leaders, users and influencers. The consortium groups more than 5,000 participants representing about 600 organizations and individuals from more than 65 countries worldwide [12].

In short, TOSCA can be used to define the interoperable description of services and applications hosted on the cloud and elsewhere, including their components, relationships, dependencies, requirements, and capabilities. It enables portability and automated management across different cloud providers regardless of underlying platform or infrastructure.

[13] http://docs.oasis-open.org/tosca/TOSCA/v1.0/os/TOSCA-v1.0-os.html
[14] www.oasis-open.org

The TOSCA specification provides a language to describe both:

- Service components and their relationships using a service topology, and;
- The management procedures that create or modify services using orchestration processes.

According the specification these definitions are packaged in a so called "Cloud Service ARchive" (CSAR). The following figure illustrates the whole idea:

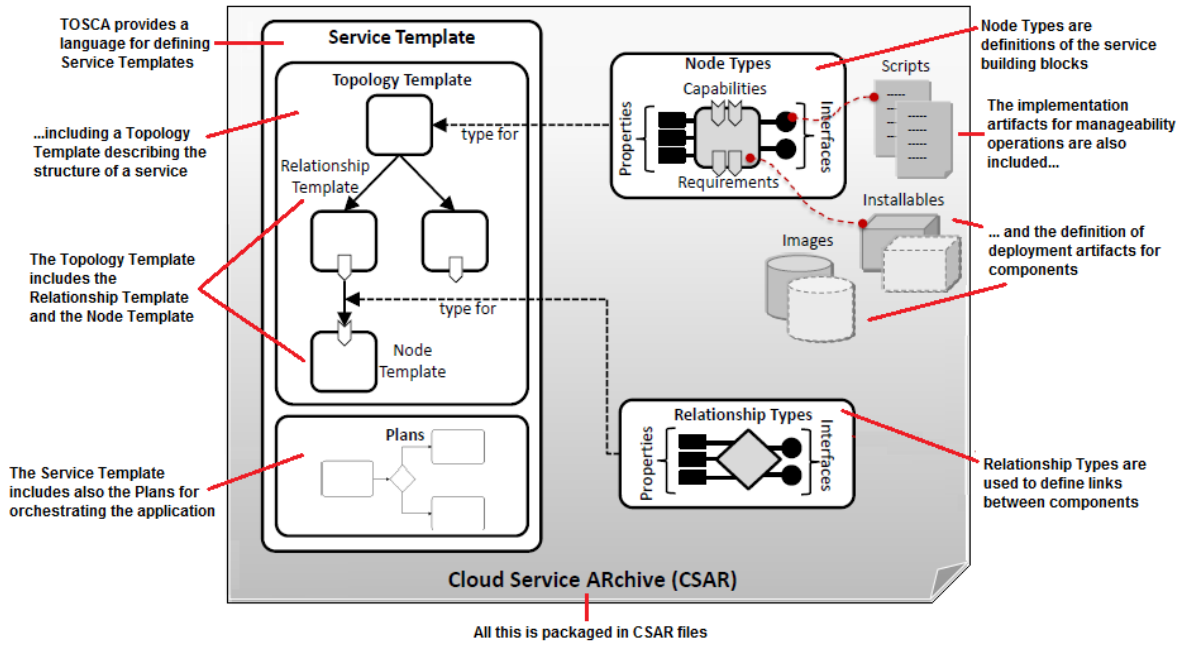

Figure 5. TOSCA CSAR package

The CSAR is a container file, i.e. it contains multiple files with possibly different file types. These files are typically organized in several subdirectories, each of which contains related files (and possibly other subdirectories). The organization into sub-directories and their content is specific for a particular cloud application. CSARs are zip files, typically compressed.

An orchestration engine processing a TOSCA service template uses the lifecycle operations to instantiate single components at runtime, and it uses the relationship between components to derive the order of component instantiation. For example, during the instantiation of an application including a web server depending on a database, an orchestration engine would first invoke the 'create' operation on the database component to install and configure the database itself, and it would then invoke the 'create' operation for the web server to install and configure it.

3.2.2 Language Overview

TOSCA is good when it comes to defining virtual application topologies, VNF dependencies and relationships and actions to be performed as part of a lifecycle. To get this, the TOSCA language introduces a grammar for describing service templates by means of two main building blocks: *Topology Templates* and *Plans*[15].

[15] The complete EBNF grammar can be found in the TOSCA specification document (Appendix C).

3.2.2.1 Topology Templates and Plans

The TOSCA specification defines a *meta-model* for defining IT services. This meta-model defines both:

- The structure of a service (by means of *Topology Templates*), and;
- How to manage it (by means of *Plans*).

Both are defined in the so called *Service Template*.

A *Topology Template* (also referred to as the *topology model* of a service) defines the *structure* of a service.

For example, let us consider a service consisting on:

- An application server,
- A process engine, and;
- A process model.

A Topology Template defining that service would include different *Node Templates*:

- A Node Template of Node Type “application server”,
- Another Node Template of Node Type “process engine”, and;
- A third Node Template of Node Type “process model”.

The application server *Node Type* will define properties such:

- The IP address of an instance,
- an operation for installing the application server with the corresponding IP address, and
- an operation for shutting down an instance of this application server.

A constraint in the Node Template could specify a range of IP addresses available when making a concrete application server available.

Additionally, a *Relationship Template* can be used to specify the occurrence of a relationship between nodes in a Topology Template. Each Relationship Template refers to a Relationship Type that defines the semantics and any properties of the relationship. The Relationship Template indicates the elements it connects and the direction of the relationship by defining source and target elements.

In our example, a relationship could be established between the process engine Node Template and application server Node Template with the meaning “hosted by”, and between the process model Node Template and process engine Node Template with meaning “deployed on”.

On the other hand, *Plans* are used to define the process models that are used to create and terminate a service, as well as to manage a service during its whole lifetime. They are defined in the Service Template describing the management aspects of service instances, especially their creation and termination. These plans are defined as process models, i.e. a workflow of one or more steps.

3.2.2.2 Sample Template

All elements needed to define a TOSCA Service Template – such as Node Type definitions, Relationship Type definitions, etc. – as well as Service Templates themselves are provided in the so called *TOSCA Definitions documents*. These TOSCA Definition documents are based on the XML Schema 1.0 specification.

Just as an example, the following shows a sample Definitions document defining two Node Types (*Application* and *ApplicationServer*) and a Relationship Type (*ApplicationHosted OnApplicationServer*).

```
<Definitions id="MyDefinitions" name="My Definitions"
  targetNamespace="http://www.example.com/MyDefinitions"
  xmlns:my="http://www.example.com/MyDefinitions">
```

```
        <Import importType="http://www.w3.org/2001/XMLSchema"
          namespace="http://www.example.com/MyDefinitions">

        <NodeType name="Application">
          <PropertiesDefinition element="my:ApplicationProperties"/>
        </NodeType>

        <NodeType name="ApplicationServer">
          <PropertiesDefinition element="my:ApplicationServerProperties"/>
        </NodeType>

        <RelationshipType name="ApplicationHostedOnApplicationServer">
          <ValidSource typeRef="my:Application"/>
          <ValidTarget typeRef="my:ApplicationServer"/>
        </RelationshipTemplate>

      </Definitions>
```

3.2.3 TOSCA and NFV

OASIS, the standards organization responsible for TOSCA, has a technical committee draft document providing a specific profile for NFV that was recently empowered to consider how NFV standardized models can describe Network Services composed of Virtual Network Functions and, *in turn*, composed of Virtualised Deployment Units using network traffic flow descriptors (the so-called Forwarding Graphs) [13]. In order to do this, they are working closely with the NFV community. For example, the open source OpenStack Tacker[16] (NFV Orchestration) project is working to use actual TOSCA Service Templates as input to deploy virtual Network Routers, Switches and Multi-Function devices.

[16] https://wiki.openstack.org/wiki/Tacker

Figure 6 shows the outline for the definition of a NS in TOSCA. The first elements (definition of VNFs, VLs and CPs) establish the Network Connectivity Topology (NCT) of the NS, while the other two elements (VNFFPs and VNFFGs) determine the possible data paths over the NCT.

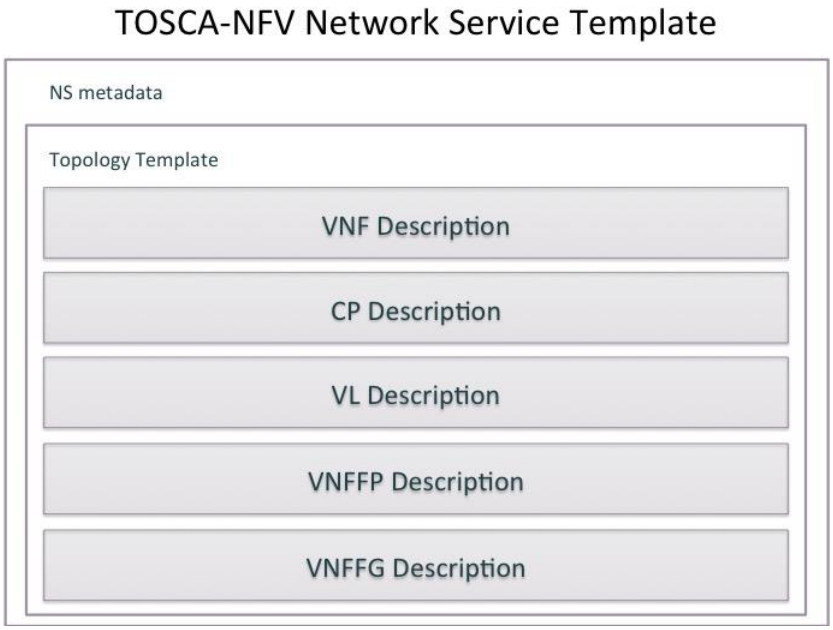

Figure 6: TOSCA Template for Network Service Description

A Network Service Descriptor (NSD) describes the attributes and requirements necessary to create a NS. In this document, metadata elements are used to define service specific properties, such as ID, vendor or version.

```
tosca_definitions_version:      tosca_simple_profile_for_nfv_1_0
tosca_default_namespace:    # Optional. default namespace (schema, types version)
description: example for a NSD.
metadata:
    ID:                                 # ID of this Network Service Descriptor
    vendor:                             # Provider or vendor of the Network Service
    version:                            # Version of the Network Service Descriptor
imports:
    - tosca_base_type_definition.yaml
    # list of import statements for importing other definitions files
topology_template:
    inputs:
            flavor ID:
VNF1:
            type: tosca.nodes.nfv.VNF.VNF1
            properties:
```

Next, the Topology Template comprises descriptors for all the elements required to configure a NS (VNFs, VLs, CPs, VNFFPs and VNFFGs).

Each VNF is considered as a subsystem in a NS and is described as a node template with the needed requirements, each for a different VL:

```
topology_template:
    inputs:
            flavour ID:
VNF1:
            type: tosca.nodes.nfv.VNF.VNF1
            properties:
                Scaling_methodology:
                Flavour_ID:
                Threshold:
                Auto-scale policy value:
                Constraints:
            requirements:
                virtualLink1: VL1
                virtualLink2: VL2
                virtualLink3: VL3
            capabilities:
                forwarder1
                forwarder2
                forwarder3

     VNF2:
            type: tosca.nodes.nfv.VNF.VNF2
            properties:
```

Connection Points act as the external endpoints of the Network Service:

```
                forwarder2
                forwarder3

       CP01          #endpoints of NS
            type: tosca.nodes.nfv.CP
            properties:
                type:
            requirements:
                virtualLink: VL1

       CP02        #endpoints of NS
            type: tosca.nodes.nfv.CP
            properties:
```

Virtual Links are described as node templates representing a logical virtual link entity:

```
           requirements:
                virtualLink: VL4


      VL1
            type: tosca.nodes.nfv.VL.Eline
            properties:
                  # omitted here for brevity
             capabilities:
                  -virtual_linkable
                  occurrences: 2

      VL2
            type: tosca.nodes.nfv.VL.ELAN
            properties:
```

A Network Forwarding Path is an ordered list of Connection Points that form a chain of VNFs. The order of network functions applied is application-dependent, and may be a simple sequential set of functions, or a more complex graph with alternative paths (e.g., the service may fork, and even later combine), depending on the nature of the traffic, the context of the network, and other factors.

```
       occurrences: 2

      Forwarding path1:
            type: tosca.nodes.nfv.FP
            description: the path (CP01-CP11-CP13-CP21-CP31-CP33-CP02)
            properties:
               policy:
            requirements:
               -forwarder: CP01
               -forwarder: VNF1
                 capability: forwarder1         #CP11
               -forwarder: VNF1
                 capability: forwarder3         #CP13
               -forwarder: VNF2
                 capability: forwarder1         #CP21
               -forwarder: VNF3
                 capability: forwarder1         #CP31
               -forwarder: VNF3
                 capability: forwarder3         #CP33
               -forwarder: CP02

      Forwarding path2:
            type: tosca.nodes.nfv.FP
            description: the path (CP01-CP11-CP13-CP31-CP33-CP02)
            properties:
```

The NFV VNFFG group type represents a logical VNF forwarding graph entity. A VNFFG is specified by a Network Service Provider to define how traffic matching certain criteria (defined by the policy element) is intended to flow through one or more network functions in a NCT in order to accomplish the desired network service functionality. The NFV specification describes VNFFGs using one or more Network Forwarding Paths:

```
               -forwarder: VNF3
                 capability: forwarder3         #CP33
               -forwarder: CP02

      Groups:
          VNFFG1:
               type: tosca.groups.nfv.vnffg
              description: forwarding graph 1
              properties:
                vendor:
                version:
                vl: [VL1,VL2,VL4]
                vnf: [VNF1,VNF2,VNF3]
              targets: [Forwarding path1, Forwarding path2]

          VNFFG2:
              type: tosca.groups.nfv.vnffg
```

```
description: forwarding graph 2
```

Typically TOSCA is used in NFV to solve four different problems:

- To describe the topology of a network service or VNF as defined by ETSI NFV (defining node templates to describe components in the topology structure, and then, defining relationship templates to describe connections, dependency and deployment order).
- ***Composition***. Using the TOSCA substitution feature the NFV information model can be described by using multiple TOSCA service templates. Also, any node in a TOSCA topology can be an abstraction of another layer or sub-topology.
- ***Lifecycle control***. TOSCA models have a consistent view of state-based lifecycle. By default, it defines *operations* (such start, stop, create, configure…) which can be sequenced against the state of any dependent resources; also, additional ad-hoc operations can be defined when necessary (such *post_start*, *post_configure*, etc.). It allows handling complex state changes, configurations, etc. As a whole, the lifecycle can be customized for the specific NFV resources and relationships.
- ***Portability***. TOSCA makes possible a flexible movement between different cloud infrastructures for NFV applications by expressing the application requirements independently from cloud capabilities and implementations. It provides multi VIM support and also allows manipulating the orchestration declaratively instead of dealing with different cloud APIs.

3.2.4 Related Projects

3.2.4.1 Cloudify

Cloudify [14] is an open source TOSCA-*based* cloud orchestration framework written in Python[17], and YAML. It was created by GigaSpaces Technologies and licensed under the Apache License Version 2.0[18]. Basically, Cloudify is used to automate the process of installation, deployment and also post-deployment such as application monitoring, remediation, and auto-scaling.

Cloudify uses the TOSCA specification as its standard templating language. The current integration includes mapping of the current HOT template into the Cloudify/TOSCA format.

Cloudify acts as a "software operator" that follows similar steps to the way a human operator works, but in a fully automated manner. To get to this level of automation Cloudify receives input that can be understood by software configuration files, known as "blueprints", which describe how the application interacts with the data centre through, to execute the desired blueprint configurations.

The typical steps for deploying and managing an application using Cloudify are:

- Describe the application in a TOSCA-based blueprint, which includes the details required to install and manage the application (the components and their dependencies, the location of the binaries and the configuration for the installation and monitoring processes).
- Execute the plan: Cloudify uses the blueprint as input describing the deployment plan and is able to execute it on a variety of cloud environments.

Cloudify supports a variety of cloud platforms (including OpenStack, AWS, VMWare[19] and others), but also is supported on Non-Cloud Platforms (Bare Metal[20]) by plugging in to a predefined pool of machines and managing the application deployment within the boundaries of that pool. This configuration is commonly used on environments without a cloud-based

[17] https://www.python.org/
[18] For more information, see: http://www.apache.org/licenses/
[19] http://www.vmware.com/
[20] https://en.wikipedia.org/wiki/Bare_machine

environment, or for applications that are performance and I/O intensive such as certain Network Functions.

3.2.4.2 Alien4Cloud

Alien4Cloud (*Application LIfecycle ENabler for Cloud*) [15] is an open source TOSCA-*based* designer and Cloud Application Lifecycle Management Platform. Basically, it is an application that allows people in the enterprise to collaborate in order to provide self-service deployment of complex applications. It makes possible to deploy complex applications and leverage cloud resources to setup their environments in minutes.

This project was started to help enterprises adopting the cloud for their new and existing applications in an Open way. The code is written in Java and it has been developed using an Open-Source model (Apache 2 License).

In order to provide an enterprise self-service portal, Alien4Cloud leverages the following concepts:

- **Location**: Deployment target (cloud or set of physical machines).
- **Components**: Software components to deploy.
- **Topologies** (or blueprints): Description of multiple software components assembled together (to build an application).
- **Applications**: Actual applications to deploy with environments and versions each of them being associated with a topology.
- **TOSCA**: To describe service components and their relationships.

In Alien4Cloud, TOSCA can be used to define both Types (catalogue elements) and Applications topologies (Templates). Alien4Cloud tools -like the topology editor- allows you to create Application topologies that can be exported to TOSCA Templates.

Alien4Cloud support a slightly modified version of TOSCA Simple Profile in YAML, in order to add features that are specific to Alien4Cloud context. However, it is also able to load pure TOSCA compliant templates and also export topologies as pure TOSCA templates.

Alien4Cloud provides the following capabilities:

- Ease the design and portability of Applications using TOSCA.
- Isolate the application evolution from deployment technologies and infrastructures, allowing to integrate with any deployment layer and infrastructure.
- Accelerate Application Infrastructure Design and improve reusability by providing a Components and Blueprints catalogues.
- Ease collaboration between Development and Deployment teams across the Application lifecycle in creating the Components and Blueprints to fill the catalogue.
- Integrate with existing Enterprise systems (Dev and Ops) through a REST API and pluggable strategies.

3.2.4.3 OpenStack HEAT

As mentioned, the OpenStack Foundation has also defined a similar standard for specifying resources and the orchestrations for managing infrastructure and application lifecycles called HEAT (*section* 4.1 above).

As part of HEAT, the TOSCA-to-HEAT translator project was one of the first to adopt TOSCA for standardized templating, so automated TOSCA-*based* orchestration is now part of OpenStack. This translator receives a TOSCA template as input and generates a HEAT Orchestration Template (a HOT file) as output.

3.2.4.4 Tacker

Tacker is a project incubated inside OpenStack to develop a general-purpose network functions virtualisation orchestrator. The purpose is to build a Generic VNF Manager (VNFM) and a NFV Orchestrator (NFVO) according the ETSI MANO specification [1]. It provides a functional stack to

Orchestrate Network Services end-to-end () using VNFs (i.e.: taking care of configuring the VNF, monitoring it, and if needed, rebooting and/or scaling when necessary).

The relation with TOSCA is because Tacker uses TOSCA for VNF meta-data definition. One of the main components of Tacker is the VNFD Catalogue (which stores the VNF descriptors) and the objects stored there are TOSCA templates.

Once a VNF has been described using the TOSCA NFV template it can be on-boarded into the Tacker VNF Catalogue (the VNF Catalogue is one of the main components of Tacker). Once on-boarded, Tacker can instantiate the VNF by interpreting the TOSCA template and translating appropriate portions to OpenStack Heat using a translator.

3.3 YANG

3.3.1 Introduction

YANG (acronym for "*Yet Another Next Generation*") is a data modelling language for the network configuration protocol NETCONF[21]. NETCONF is commonly used to configure and interact with both, physical and virtual network devices such as routers, firewalls and switches. NETCONF is considered by many as the potential successor of the well-known SNMP (Simple Network Management Protocol) [3].

YANG was developed by the NETMOD working group[22] in the IETF and published as the RFC6020 in October 2010 [16]. It can be used to model both: (i) configuration data, and; (ii) state data manipulated by the Network Configuration Protocol (technically, it is used to model the 'operations' and 'content' layers in the NETCONF protocol)[23]. Additionally, it can be also used to define the format of event notifications emitted by network elements.

YANG models are human readable and NetCONF/YANG interactions meet a range of criteria to provide safe provisioning, with lock, unlock, commit and rollback functionality. Also, YANG comes with a number of built-in data types, but additional application specific data types can be derived from those built-in data types.

YANG maintains compatibility with Simple Network Management Protocol's (SNMP's) SMIv2 (Structure of Management Information version 2[24]). SMIv2-*based* MIB modules can be automatically translated into YANG modules for read-only access. However, YANG is not concerned with reverse translation from YANG to SMIv2.

The YANG main strength is for configuring network devices. As a whole, it is considered a very good resource for configuring network devices and network services in a human readable fashion.

3.3.2 Language Overview

YANG models the hierarchical organization of data as a tree in which each node has a name, and either a value or a set of child nodes. YANG provides clear and concise descriptions of the nodes, as well as the interaction between those nodes.

YANG structures data models into modules and submodules. The module is the base unit of definition in YANG. A module defines a single data model. Submodules are partial modules that

---

[21] The Network Configuration Protocol is a network management protocol developed and standardized by the IETF (RFC 4741 and RFC 6241). It provides mechanisms to install, manipulate, and delete the configuration of network devices using RPC and XML as the data encoding language.

[22] See, *for example*: https://trac.tools.ietf.org/wg/netmod/trac/wiki

[23] The NETCOF protocol can be conceptually partitioned into four layers: Content, Operations, Messages and Secure Transport. See RFC 4741 for details.

[24] For more details, see: https://tools.ietf.org/html/rfc2578

contribute definitions to a module (a module may be divided into submodules). A module may include any number of submodules, but each submodule may belong to only one module.

A module can import data from other external modules, and include data from submodules. As a whole, a YANG module allows a complete description of all data sent between a NETCONF client and server.

A module contains three types of statements:

- module-header statements (describes the module and give information about the module itself);
- revision statements (gives information about the history of the module);
- definition statements (the body of the module where the data model is defined).

On the other hand, YANG defines different types of nodes for data modelling:

- Leaf Nodes, containing simple data like an integer or a string.
- Leaf-list. This is a sequence of leaf nodes with exactly one value of a particular type per leaf
- Container node. This is used to group related nodes in a subtree. A container has only child nodes and no value. A container may contain any number of child nodes of any type (including leafs, lists, containers, and leaf-lists).
- A list defines a sequence of list entries. Each entry is like a structure or a record instance, and is uniquely identified by the values of its key leafs. A list can define multiple key leafs and may contain any number of child nodes of any type (including leafs, lists, containers, etc.).

The following shows a YANG sample containing the mentioned data types[25]:

```
module acme-system {
    namespace "http://acme.example.com/system";
    prefix "acme";

    organization "ACME Inc.";
    contact "joe@acme.example.com";
    description
        "The module for entities implementing the ACME system.";

    revision 2007-06-09 {
        description "Initial revision.";
    }

    container system {
        leaf host-name {
            type string;
            description "Hostname for this system";
        }

        leaf-list domain-search {
            type string;
            description "List of domain names to search";
        }

        container login {
            leaf message {
                type string;
                description
                    "Message given at start of login session";
            }
```

[25] This has been taken from the YANG language specification itself.

```
                list user {
                    key "name";
                    leaf name {
                        type string;
                    }
                    leaf full-name {
                        type string;
                    }
                    leaf class {
                        type string;
                    }
                }
            }
        }
    }
```

YANG can model state data as well as configuration data, based on the "config" statement. When a node is tagged with "config false", its subhierarchy is flagged as state data, to be reported using NETCONF's <get> operation, not the <get-config> operation. Parent containers, lists, and key leafs are reported also, giving the context for the state data.

In the following example two leafs are defined for each interface, a configured speed and an observed speed. The observed speed is not configuration, so it can be returned with NETCONF <get> operations, but not with <get-config> operations. The observed speed is not configuration data, and it cannot be manipulated using <edit-config>:

```
    list interface {
        key "name";

        leaf name {
            type string;
        }
        leaf speed {
            type enumeration {
                enum 10m;
                enum 100m;
                enum auto;
            }
        }
        leaf observed-speed {
            type uint32;
            config false;
        }
    }
```

YANG modules can be translated into an alternative XML-*based* syntax called YIN[26]. The translated module is called as a YIN module.

The YANG and YIN formats contain equivalent information using different notations. The YIN notation enables developers to represent YANG data models in XML and, therefore, use the rich set of XML-*based* tools for data filtering and validation, automated generation of code and documentation, and other tasks. Tools like XSLT or XML validators can be utilized. The mapping between YANG and YIN does not modify the information content of the model (comments and whitespace are not preserved).

[26] See, *for example*: https://tools.ietf.org/html/rfc6020

For instance, the following YANG module:

```
 module acme-foo {
    namespace "http://acme.example.com/foo";
    prefix "acfoo";

    import my-extensions {
        prefix "myext";
 }

    list interface {
        key "name";
        leaf name {
            type string;
        }

        leaf mtu {
            type uint32;
            description "The MTU of the interface.";
            myext:c-define "MY_MTU";
        }
    }
}
```

can be translated into the following YIN code:

```
<module name="acme-foo"
        xmlns="urn:ietf:params:xml:ns:yang:yin:1"
        xmlns:acfoo="http://acme.example.com/foo"
        xmlns:myext="http://example.com/my-extensions">

  <namespace uri="http://acme.example.com/foo"/>
  <prefix value="acfoo"/>

  <import module="my-extensions">
    <prefix value="myext"/>
  </import>

  <list name="interface">
    <key value="name"/>
    <leaf name="name">
      <type name="string"/>
    </leaf>
    <leaf name="mtu">
      <type name="uint32"/>
      <description>
        <text>The MTU of the interface.</text>
      </description>
      <myext:c-define name="MY_MTU"/>
    </leaf>
  </list>
</module>
```

3.3.3 YANG and NFV

In the field of SDN and NFV, YANG is a popular data modelling language that serves as a contractor between network devices and those that interact and interface with these devices. The YANG data model helps to define how to write configuration data, as well as how to communicate state data in hierarchical manner that is strongly typed.

When it comes to NFV orchestration, the orchestrator needs to work with different NFV components, such as SDN controllers (e.g. OpenDaylight) as well as physical and virtual network functions (i.e., load balancers, routers, switches, firewalls). YANG is often the language of choice to model and configure these devices.

3.3.4 Related Projects

Open Source MANO (OSM) is an ETSI-*hosted* project to develop an Open Source NFV Management and Orchestration (MANO) software stack aligned with ETSI NFV (https://osm.etsi.org/).

The project was launched in the first quarter of 2016 and it has currently the Release 0 of the software, which provides a solid platform on which to develop the new features to be included in Release 1.

The adopted OSM data model is purely based on the ETSI NFV information model. A description can be found in their website[27]. The specific details of the modelling language and representation model are also available online[28].

As can be read, OSM adopts YANG as the modelling language and thus ETSI MANO objects are modelled as YANG objects.

These data model is then represented in yaml format to create descriptor files (nsd.yaml, vnfd.yaml).

3.4 Model-extending languages: example of T-Nova descriptor

The formerly described languages provide standard-*based* languages (or well-adopted *de-facto* standard in case of OpenStack) to implement the ETSI NFV data model. However, different projects also provide their specific solutions to formally describe the ETSI NFV data model also incorporating the required elements to support their proposed extensions.

A good example is the European FP7 T-NOVA project, which provides an open source implementation based on the ETSI NFV model.

T-NOVA extends the information model proposed by ETSI NFV adding relevant fields from the proposed marketplace point of view, as well as applying the TMForum SID model [17] to the model provided by ETSI NFV.

Based on the above two main standardization bodies active in defining information models for NFV, the T-NOVA project has taken the two models in order to apply business aspects from TMForum SID model[29], to the model proposed by ETSI NFV. This approach is depicted in Figure 7

[27] https://osm.etsi.org/wikipub/index.php/Release_0_Data_Model_Details
[28] https://osm.etsi.org/wikipub/images/8/8e/OSM_Release0_Data_Models.pdf
[29] https://en.wikipedia.org/wiki/Information_Framework_(Frameworx)

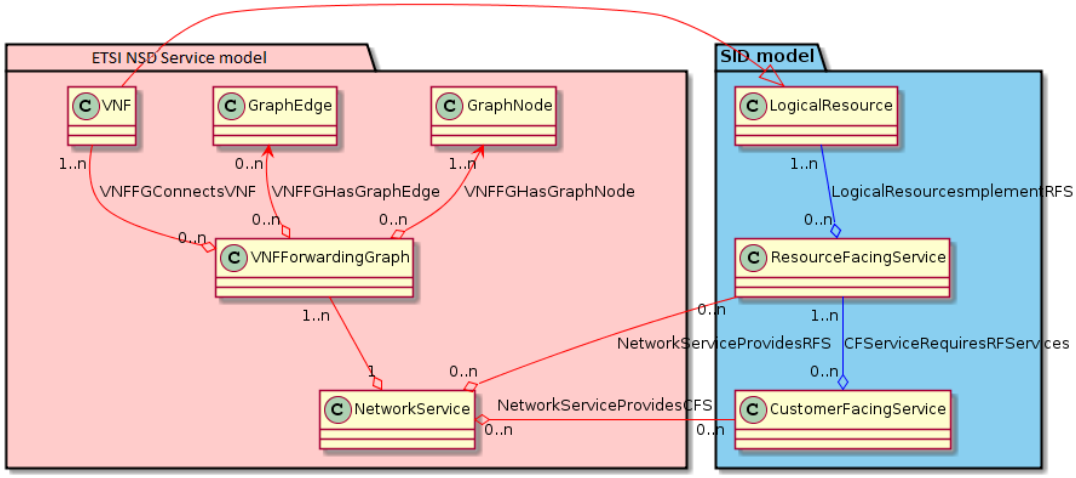

Figure 7: Applying SID service model to ETSI NFV information model

Taken ETSI NFV Network Service Descriptor (NSD) and Virtual Network Function Descriptor (VNFD) as the baseline, T-NOVA has extended them by adding several fields needed according to the requirements defined for T-NOVA Marketplace. Also other fields have been modified or added in T-NOVA from the VNF and Network Services operational point of view as they are managed by the orchestrator [18], [19].

3.4.1 T-NOVA VNFD

The T-NOVA Virtual Network Function Descriptor (VNFD) is the template which describes a VNF in terms of deployment and operational behaviour requirements. The VNFD also contains connectivity, interface and KPIs requirements that can be used by NFV-MANO functional blocks to establish appropriate Virtual Links within the NFVI between VNFC instances, or between a VNF instance and the endpoint interface to other Network Functions.

The T-NOVA VNFD also includes all the information that will allow the commercial activity of VNF through a marketplace, such vendor/owner, SLA, billing models, price, etc.

Figure 8 represents T-NOVA VNFD information components relevant to the marketplace and their relation.

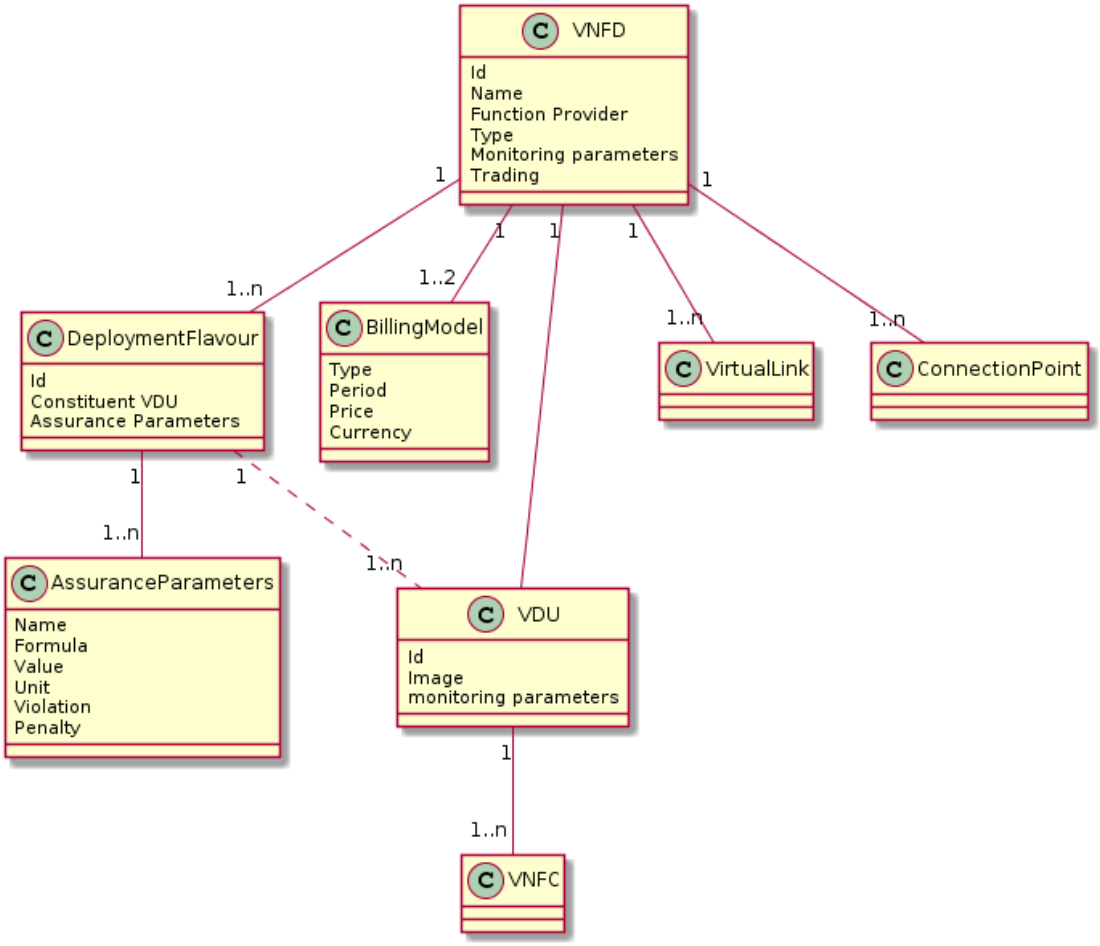

Figure 8: T-NOVA VNFD information model

The whole T-NOVA VNFD can be found in [20]. The VNFD Preamble (upper box) provides the necessary information for the release, id, creation, provider etc. In fact, T-NOVA extends the information with Marketplace-*related* information such as trading and billing. The VDU VNFD segment provides information about the required resources that will be utilised in order to instantiate the VNFC. The configuration of this part may be extremely detailed and complex depending on the platform specific options that are provided by the developer. However, it should be noted that the more specific the requirements are stated here, the less portable the VNF might be, depending on the NFVO policies and the SLA specifications. It is assumed that each VDU describes effectively the resources required for the virtualisation of one VNFC. The VNFC subsection is related to the internal structure of the VNF and describes the connection points of the VDU (name id, type) and the virtual links where they are connected.

3.4.2 T-NOVA NSD

The T-NOVA NSD represents the reference data model used by the orchestrator and the marketplace. Figure 9 represents the T-NOVA NSD information components and their relation.

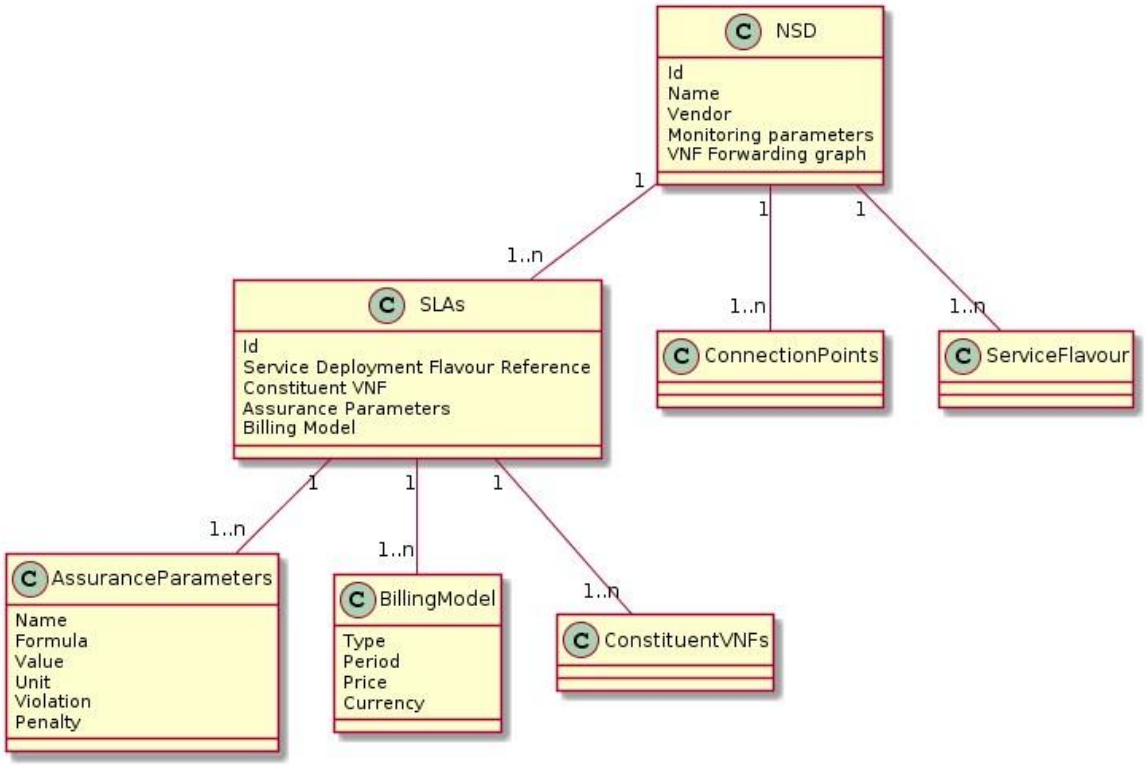

Figure 9: T-NOVA NSD information model

In T-NOVA, the NSD must include also the information that will allow to carry out the commercial activity realted to the VNFs through the marketplace, such vendor/owner, SLA, billing models, price, etc. As a consequence, many fields have been added in the T-NOVA NSD, on top of the NSD defined in [1]. A detailed list of the additional T-NOVA parameters can be found in [20], Annex B, pages 59-62.

3.4.3 T-NOVA VNF lifecycle

The VNF lifecycle represents the different stages that a VNF needs to pass through. According to [1], each VNF must have a management interface, i.e. the T-Ve-Vnfm interface, which is used to support the VNF lifecycle.

In the T-NOVA project, the VNF lifecycle includes the following stages [18]:

- **Development:** the Software implementation of Network Functions (NFs) is performed by the Function Providers. NFs are published and aggregated in the T-NOVA Function Store.
- **Validation:** the validation procedure aims at providing a certification that the developed NFs will work as expected.
- **Publication:** NF publication is performed at the NF Store, whose repository will host both the function image (as stand-alone application or integrated VM) and the associated description/metadata.
- **Brokerage:** Brokerage is undertaken by the brokerage module in the marketplace that will match users' service requirements with the technical capabilities provided by the NFs, thus ensuring that the resources required for NFs deployment are available.
- **Selection:** Finally the customer selects the most suitable NFs according to his or her needs.

- **Deployment:** The VM image and its metadata are transferred from the NF Store.
- **Management:** This is the running phase of the NF. An Application Programming Interface (API) will be exposed to the orchestrator for setup and real-time control or management. The running phase can be preceded by a network re-configuration procedure.
- **Termination:** Involves the removal of the NF instance from the virtualised infrastructure, including network re-configuration, if needed.

A simplified scheme of the T-NOVA VNF lifecycle is shown in Figure 10.

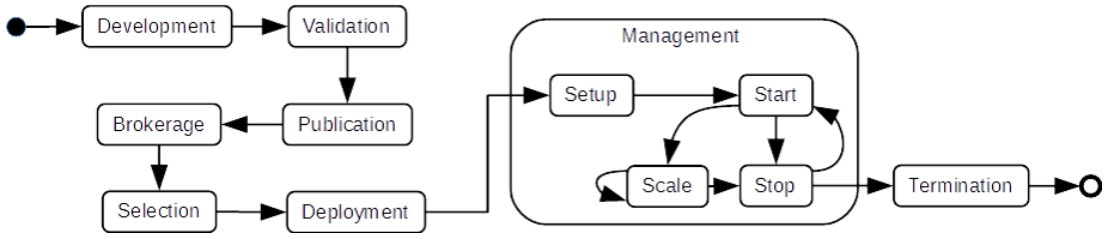

Figure 10: T-NOVA VNF lifecycle

In T-Nova, the VNF management interface is technically implemented by only one of the VNFCs constituting the VNF. Such VNFC is in charge of distributing the lifecycle-*related* information to all the others VNFCs constituting a VNF.

To this aim, each VNF needs to be able to declare various lifecycle events (e.g. start, stop, pause...). For each of those events, the information needed to configure the VNF can be very different. Moreover, the command to trigger the re-configuration of the VNF can change between events. To support the different events each one needs to be detailed in the VNFD.

The information related to the VNF life-cycle is inserted in the "lifecycle_event" section of the VNFD. In particular, in such a section the following information is available:

- **Driver**: the protocol used for the connection between the VNF Manager and the controlling VNFC. In T-Nova, two protocol can be used, the Secure Shell (ssh) protocol and the http protocol.
- **Authentication**: such fields specify the type of authentication that must be used for the connection, and some specific data required by the authentication process (e.g. a link to the private key injected by the VNFM at startup).
- **Template File Format**: specifies the format of the file that contains the information about the specific lifecycle event, and that must be transferred once the command is run.
- **Template File**: includes the name and the location of the Template File.
- **VNF Container**: specifies the location of the Template File.
- **Command**: defines the command to be executed

4 SESAME Network Function model requirements

This section focuses on the applicability of the ETSI NFV information model to the scope of the SESAME project. For this, we overview the different types of Network Functions related to the SESAME solution, identifying the most relevant parameters specific that must be taken into account in the SESAME descriptors.

4.1 Service VNF

The service VNF is devoted to running mobile edge service instances to execute virtualised service-level functions within the CESC cluster. The Service VNFs prevent the need of going through the EPC to access the required services, enabling higher responsiveness and the possibility to exploit RAN-level context information to optimise the services. Examples of Service VNFs that are considered in SESAME are: Deep packet inspection VNF (vDPI), Context aware routing VNF (vCAR), Edge caching VNF (vCaching) and Transcoding VNF (vTranscodingUnit—vTU). Detailed descriptions of service VNFs are presented in SESAME D2.2, *section 4.4* and SESAME D2.3, *section 3.2.3*.

In general, these Service VNFs can be well described with the ETSI NFV information model presented in *Section 2* of this document. The main specificity in SESAME is the use of accelerators required for achieving an accurate performance in some Service VNFs such as vTU.

4.1.1 Hardware and software accelerators

Existing modelling languages today are not able to fully describe SESAME hardware accelerators and their characteristics. For this reason, in order to enable orchestration of hardware accelerators (e.g., FPGA, GPU), there is a need to extend this modelling languages in the context of the SESAME project. More in particular the IaaS template has to be enhanced to manage, provision and define partially reconfigurable partition of the FPGA, GPU resources, etc. The Nomad[30] project, presented by VOSYS together with China Mobile and Huawei at the Austin OpenStack summit 2016 [21], aims to enable accelerator lifecycle management in OpenStack interacting with the existing OpenStack modules depicted in Figure 2. For this reason Nomad new resource types will be added to the modelling language. For instance, in the case of HEAT OS::Nomad::FPGA and OS::Nomad::GPU will be defined with properties like VM_owner, acceleration_function. Similarly, these types of attributes must be covered in the SESAME VNF:VDU descriptors.

Another group of use cases where hardware accelerators need to be described in configuration models, is when certain functionalities are offloaded from the virtual switch to specialized hardware. For instance, one might want to create an IPSec tunnel between virtual machines on different compute nodes. Tunnel endpoints can be handled by the virtual switch while encryption operations can be entirely performed on a dedicated IPSec chip. In the same manner, hardware can be used for offloading VXLAN encapsulation/decapsulation. These use cases imply that HOT templates need to be extended with new Neutron types according to the specifics of the accelerators in question, and SESAME descriptors also need to reflect these specific configuration requirements.

4.2 Small Cell VNF

The SC VNFs are devoted to deploy certain logic of the Small Cell on the NFVI as Virtualised Network Functions.

[30] https://wiki.openstack.org/wiki/Nomad

Each micro server has a single SC-Common VNF to provide a coordinating role and a SC VNF for each Tenant (i.e. VSCNO) as illustrated below. The Small Cell (SC) VNF contains the virtualised functions instantiated into the Light DC defined by SESAME related to the Radio Resource Management (RRM) and the virtualised protocol stack including the control and the data plane. The RRM layer executes several functions like the radio bearer control, the radio admission control, cell activation, connection mobility control or the inter-cell CoMP. The control plane of the protocol stack manages several interfaces: the network management, the S1 and the X2[31]. The network management interface defined by the Broadband Forum used the TR-69 protocol[32] with the TR-196 data model[33]; the S1 interface defined between the EPC and the SC, even a HeNB-GW[34] is used in between and; the X2 interface used between SCs. The X2 interface has to be used, at least, for integrating the SESAME SCs into existing networks being compatible with current macros and SCs. Finally, the SC VNF data plane contains the protocol stack where the user data from the EPC ends. The current LTE protocol stack may be split where some layers are deployed into the SC VNF and the others are deployed into the SC PNF. The protocol stack is SW-*based* unless the PDCP ciphering functionality[35]. Below, Figure 11shows the SC VNFs proposed by SESAME into the Light DC

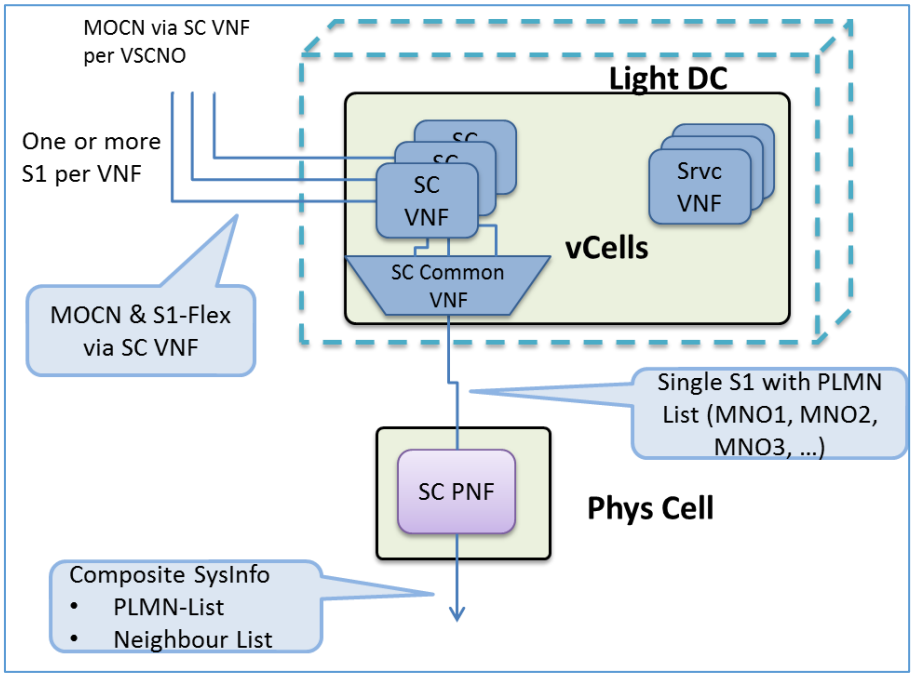

Figure 11: SESAME SC VNF

From a first general analysis of the SESAME Proof of Concept (PoC) Small Cells, it can be concluded that the SC VNFs do not require any new or special attribute not defined in the current ETSI NFV information model. With the target functional split specified in SESAME D2.3, the SC VNF can be deployed in the same way that a Service VNF is deployed since there is no special HW support related to the radio interface.

The previous review of the SC VNF functionalities in combination with the SESAME architecture proposal put the focus on the use of IPSec and the PDCP ciphering procedures for the requirement of extra HW. From one side, the proposed architecture offers the possibility to

[31] https://en.wikipedia.org/wiki/System_Architecture_Evolution
[32] https://en.wikipedia.org/wiki/TR-069
[33] https://en.wikipedia.org/wiki/TR-196
[34] See: https://en.wikipedia.org/wiki/Home_eNodeB; also see: https://en.wikipedia.org/wiki/EnodeB
[35] https://en.wikipedia.org/wiki/PDCP

establish an IPSec tunnel between the SC VNF and the tenant's EPC. From the other side, even the PoC defines the PDCP layer into the SC PNF, in case it is virtualised, the PDCP control and the data plane has to cipher the air interface. In both cases, the specific requirements need the use of HW accelerators as described in the previous section.

4.3 Physical Network Function

According to SESAME deliverable D2.3 *section 3.2.1*, the SC-PNF is a standard LTE HeNB that is responsible for the following network functions:

- Layer 3:
  - Radio Resource Control (RRC)
  - General Packet Radio Service (GPRS) Tunnelling Protocol (GTP)
  - S1 Application Protocol (S1AP)
  - X2 Application Protocol (X2AP)
  - Stream Control Transmission Protocol (SCTP)
- Layer 2:
  - Packet Data Convergence Protocol (PDCP)
  - Radio Link Control (RLC)
  - Medium Access Control (MAC)
- Layer 1:
  - LTE L1 Physical Layer (PHY) processing
  - LTE L1 Radio Frequency (RF) transmission

- User Datagram Protocol (UDP)
- Internet Protocol (IP)

Generally speaking, the SC PNF can be considered as an external element to the SESAME NFVI since it is always the last node connecting with the UE through the air interface. This SC PNFs are managed through the typical 3GPP Management System. In that sense, the entry points to the NFVI are the Virtual Links that connect the SC PNF with the SC VNF.

One specific issue in SESAME is that different SC PNFs can provide different maximum throughput in the uplink and downlink directions. A typical SC PNF can support up to 75 Mbps in DL for a single UE in good radio signal conditions in SISO, and up to 100 Mbps or 150 Mbps in MIMO. For more modern Small Cells implementing other MIMO and Carrier Aggregation techniques, the throughputs can increase up to 1 Gbps. Therefore, the SESAME SC PNF model must represent this uplink / downlink maximum bitrate.

Previous values are the maximum offered by LTE. These values will decrease on a real environment due to the physical scenario characteristics as can be the number of SCs deployed in the area; the number of UEs, how are they distributed and the distance to the cell; the propagation characteristics considering buildings and obstacles. In any case, same conditions have to be considered if the protocol stack is located at the SC VNF previously defined.

The simplest way to model this maximum and variable radio capacity for the potential different types of Small Cells is by specifying a different type of VL for each SC PNF. Each specific SC PNF will require a specific VL to connect the external connection point, and each specific type of VL will represent the related throughput information.

4.4 Sample SESAME domain

This section depicts the applicability of the ETSI NFV information model to the SESAME domain.

Figure 12 illustrates the three levels of interaction.

On the most-left column, we represent the most relevant description elements of the ETSI NFV information model and their associated attributes. These elements can be used to define the Network Connectivity Topology of a tenant, since they include the description templates for the VNFs, VLs and VNFFGs. Each type of VNF, VL and VNF-FG considered in SESAME requires a different description template.

It must be observed that the figure does not represent the VDU description templates for the instantiation of the VNFs, which would include the specific HW requirements for the different types of VNFs.

The central column represents the description templates for the instantiation of the virtualised resources, including the different instances of VNFs, VLs and VNF-FGs. Form the general description templates, each instance creates a new deployment template and includes some extra attributes related to the specific deployment.

Finally, the right-most column depicts a sample realisation of a SESAME Network Service, as described in SESAME D6.1. As it can be observed, there is a unique NS object for the tenant, which links to the different objects of the associated VNFs, VLs and VNFFGs. In this case, the tenant has two instances of the type SC VNF and one instance of the vTU, vCache and HeNB-GW VNF type. The different VNFs are interconnected though six different VL instances, and there are five different types of VNFFGs for different types of packet flows. The specific data paths associated to the VNFFGs are specified as a fixed sequence of connection points in the NFP objects.

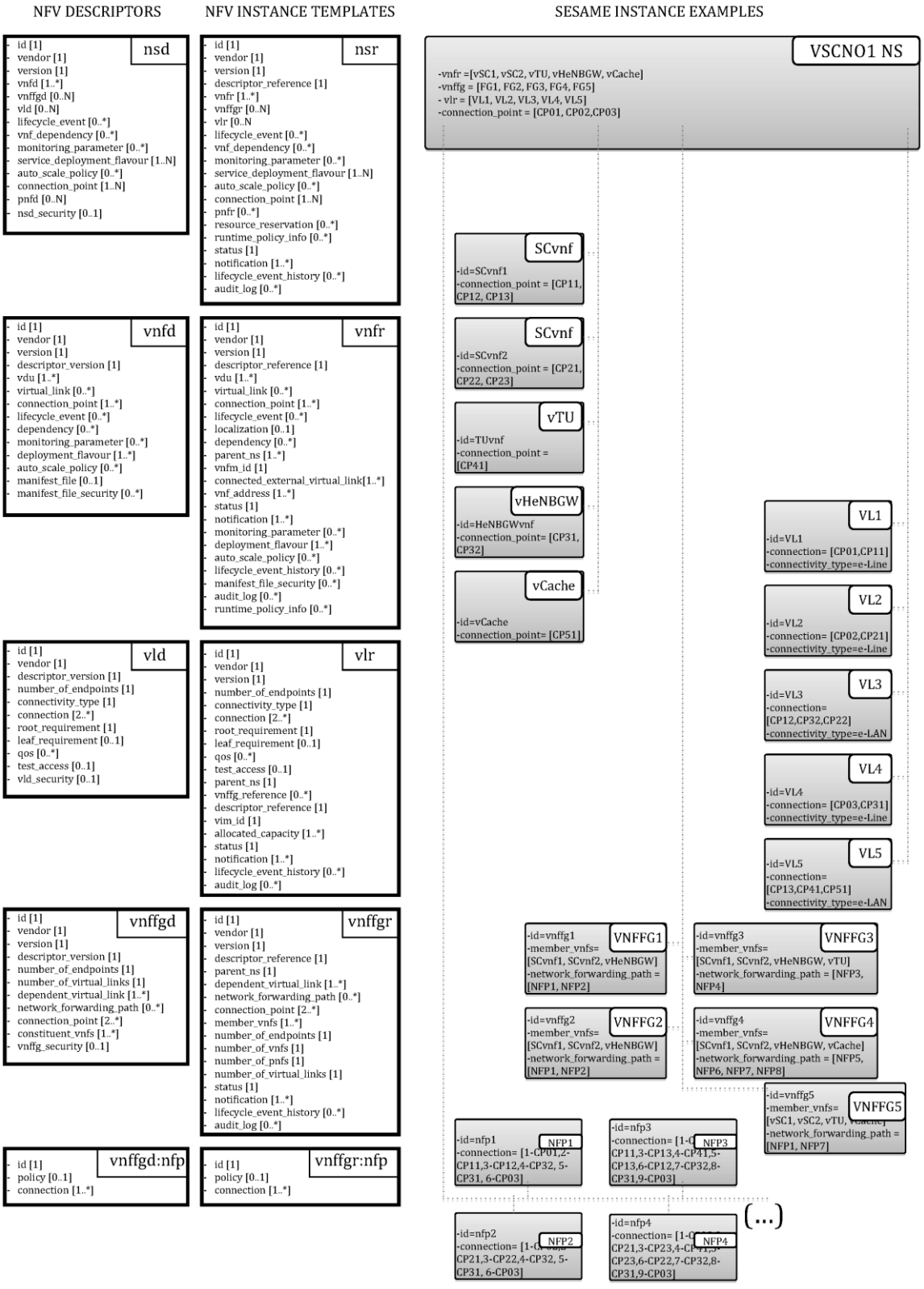

Figure 12: SESAME NFV sample

5 SESAME descriptors

In this section, the conclusions of the analysis carried out in Task 5.1, and related to the use of Descriptors in the SESAME CESC architecture will be provided. In particular, the discussion reported in the following sections is the outcome of the analysis of the available description models reported in the literature, compared to the needs and constraints brought about by architectural choices made so far by the SESAME project. Also, the experiences gained in the first stages of SESAME implementation as well as the expertise coming from partners involved in other past or ongoing projects related to Cloud Computing concepts have been taken into account.

5.1 Comparison of available solutions

In order to manage and orchestrate in an automated way VNFs within the LightDC virtualised infrastructure, the SESAME project will have to develop ad-hoc templates, so as to describe the basic components and the connectivity needed by the required network services. In particular, it is necessary to introduce the information elements described in [1], as discussed and summarised in *Section 2*. Five information elements must be necessarily used, namely the NSD, the VNFD, the PNFD, the VLD and the VNFFG. In [1], the attributes of such elements are represented by means of a number of parameters that cover a wide spectrum of possible implementations. Also, the lifecycle of the VNFs constituting the required services must be formally defined, and represented in the NSD and VNFD (for the details, see *section 2*).

The modelling of NFV components can be divided in three main categories: information models, data modelling languages and description languages.

First, in order to provide a description of the elements that are going to be used, it is necessary to define the kind of information that needs to be considered. For the NFV case, ETSI NFV provides a reference information model defining all relevant information for NFV services to be deployed. Then, a data modelling language is needed to provide a representation of the information model in a specific format, defining the grammar and a structure for it. Finally, the modelling language has to be encoded in a certain format for it to be understood by the relevant modules. There is an increased tendency in the NFV world to use YAML format due to its simplicity for both humans and machines alike, although other languages such as XML and JSON are also frequently chosen.

Table 1 below summarises the three mentioned categories and examples for each of them.

Table 1: Descriptor models and formats

INFORMATION MODEL (Parameters) ***Which information is considered?***	DATA MODEL (Schema) ***How to represent the information?***	DESCRIPTION FORMAT (Language) ***How to write the information?***
ETSI NFV model	TOSCA YANG T-NOVA SESAME ...	yaml xml json ...

In order to introduce the ETSI information elements, different modelling languages can be used. The modelling languages considered in Task 5.1 are briefly presented in *Section 4*: in particular, TOSCA, YANG, the T-NOVA descriptors and the OpenStack Heat Orchestration Templates have been analysed.

For example, the use of the TOSCA file structure to accommodate the ETSI information elements has been presented in *section 3.3*. The specific use of TOSCA in a NFV environment is discussed in details in *sub-section 3.3.3*, where practical examples are given. Guidelines to represent the

ETSI information elements in TOSCA are reported also in [1], together with some practical examples with real-life VNFs.

A comparison of the considered data modelling languages is summarised in Table 2.

Table 2: A comparison of the main modelling languages

	TOSCA	YANG	HOT	T-Nova Descriptors
Short Definition	It is a standard language built specifically for orchestration. It is effective for describing how a service should be implemented, the topology of underlying resources, or how resources may be related or dependent upon each other.	YANG is a data modelling language used to describe configuration and state information. It has been used to model networking devices and services – i.e., an object and its attributes. YANG defines the data models that are then manipulated through the NETCONF protocol.	Used to orchestrate composite cloud applications using a declarative template format through an OpenStack-native REST API.	T-NOVA extends the information model proposed by ETSI NFV adding relevant fields from the commercial point of view. It also applies the TMForum SID model [17] to the model provided by ETSI NFV.
Strengths	TOSCA's strength is orchestration itself. In other words, TOSCA can describe very well the coordination between diverse resources across a potentially complex application environment.	YANG's strength is in configuring networking devices (for example, a device such as a router can be modelled in YANG and configured via NETCONF).	Linked to OpenStack, which is the IaaS de-facto standard. Additionally HOT is the HEAT de-facto standard also (it is gradually replacing the legacy AWS CFN format)	Linked to the open source T-Nova orchestrator, which can allow significant sw re-use in SESAME.
Used to…	Used to describe orchestration (TOSCA describes both "what it is" and "what to do").	YANG is focused on describing devices (YANG provides a way for describing "what a service is").	Used to describe orchestration ("what it is" and "what to do").	Model Network Services, taking into account the commercial aspects related to the use of VNFs.

However, it should be noted that the proposed schemes can also be combined, to enhance the management capabilities of the overall framework. In [22], *for example*, the combination of TOSCA and YANG/NETCONF is suggested, in order to improve the runtime configuration capabilities of the overall system. Another meaningful example is represented by the T-NOVA descriptors, which extends the ETSI information model using the TMFORUM SID model, to add commercial information relevant to the T-NOVA Marketplace activities.

It should also be highlighted that different modelling schemes can also be used at the same time in a NFV environment. In T-NOVA, the descriptors are used mainly by the orchestrator, which then translates them into Heat Orchestration Templates (HOT), in order to forward the needed information to the T-NOVA Virtualised Infrastructure Manager, based on OpenStack.

Finally, different languages can also be used to write specific information into the descriptors. For example, T-NOVA descriptors are written in JSON; though, the use of other languages, such as XML or YAML is also possible.

5.2 Adopted solution

The analysis carried out in Task 5.1, summarisedd in the previous sections of this document, has shown that different options can be adopted by SESAME in order to introduce the main NFV concepts in the CESCs.

The fact that strict requirements for the descriptors have not been identified to be imposed in SESAME provides flexibility for deciding the type of data modelling language and description formats that are going to be used. Any of the available solutions can cover the requirements either directly or by simply making minor modifications.

An important aspect to be considered for the decision is that the choice made by the project needs to take into account the implementation solutions that are expected to be used in the project. In particular, the choice of the CESC model descriptors is mainly driven by two implementation decisions, namely the adoption of TeNOR, i.e. the T-NOVA orchestrator, as a starting point to develop the SESAME orchestration functionality, and of OpenStack as the Virtual Infrastructure Manager. Based on such choices, the descriptor schemas will be ETSI-compliant based on the ones from T-NOVA as the initial version of SESAME descriptors, slightly modified to adapt them to SESAME characteristics. Such descriptors will thus be used as the SESAME "internal" data structure, and will be used and processed by the SESAME orchestrator. Furthermore, they will be translated by the orchestrator into Heat OpenStack Templates, in order to forward the needed information from/to the VIM.

It should be noted that this solution does not put any constraint on the information models used by the VNFs providers. With the aim of taking advantage of the variety of descriptor languages in current MANO projects, SESAME will allow the use of other ETSI-compliant solutions such as TOSCA and YANG as input descriptors. To do so, the NFVO will have the capability of translating from common languages to the SESAME internal format.

5.3 Descriptor processing workflow

The descriptors are an essential part of SESAME architecture and NFV concept because they are deployment templates which describe a network service in terms of deployment and operational behaviour requirements.

Consequently, a Network Service Descriptor, in SESAME approach, can be created and sent in NFVO from two different sources. These sources can be either NMS or CESCM Portal. After receiving the NSD descriptor, NFVO component starts the process of translating the relevant information from the descriptor into a template. In SESAME, we have chosen to use HEAT OpenStack Orchestrator and HOT template. HOT template is considered reliable, supported and standardized for the OpenStack releases[36] (Juno, Kilo and Liberty) that SESAME will be deployed. Since the Network Service Descriptor is translated into a HOT template then the NFVO component will pass the translated template to VIM. Thereafter, VIM manages the NFV infrastructure and provides the management capability required to deploy/configure/destroy VNFs running in a virtual environment through the VNFI and the OpenStack modules.

[36] http://releases.openstack.org/

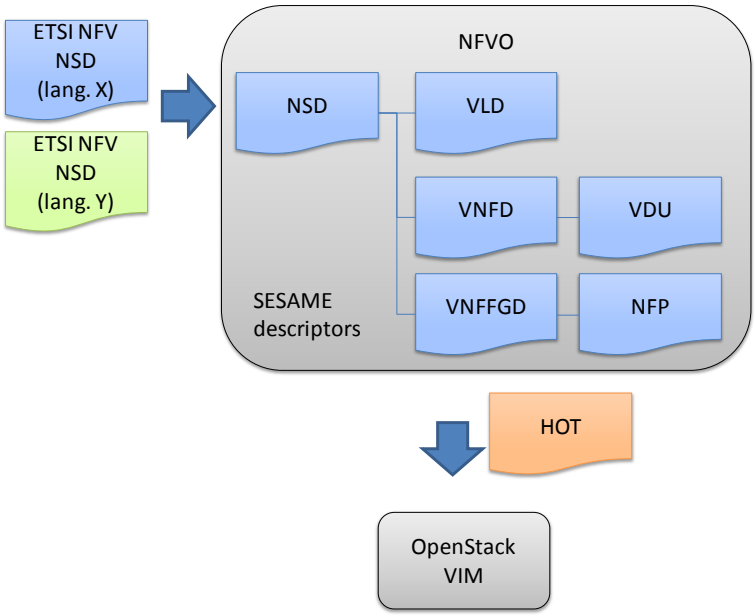

Figure 13: SESAME descriptor workflow

A NSD descriptor offers important information about the NFVI and the VNFs; the information that the subsection concerns is the following. NSD descriptor has a reference node to another descriptor, VNFD. This descriptor can refer to a variety of VNFs because several VNFs can be part of a relevant network service. Also the VNFD descriptor has a reference link to VDU; a Virtual Deployment Unit defines the compute resources of a Virtual Machine. For example it offers description about the VM image, computation requirements, virtual characteristics about the memory and network and parameters for constraint, high availability and scale in/out.

It is important to note that the VMs hosting one or multiple VNFs will be dynamic, not static in terms of a VNF may be added or stopped, according to the service and traffic requirements that arise in any given period of time. This process will probably be controlled by sending updates for the VMs through the VIM, or to the VMs directly.

Finally, we expose one last descriptor, which is necessary for the virtualisation needs of the SESAME project. We refer to VNFFG information element, which contains metadata about the VNF forwarding graph, references to VLs, VNFs and PNFs. The VNFFG can have a requirement for adding a service ID/tag to the VNFFGR and all VNF and PNF records used to fulfil the described VNFFG. This can help fingerprint/track everything related to that service in a NFVO, and will be needed to ensure that effective test access can be achieved for fault localisation.

Thus the HOT template describes a complete and totally configured virtual environment for the VNFs. NFVO passes the HOT template to the VIM component and this, in turn, updates the NFVI with VDU metadata and SDN controller with VNFFG metadata and also notifies the VNFM for the VNF lifecycle and initial configuration parameters.

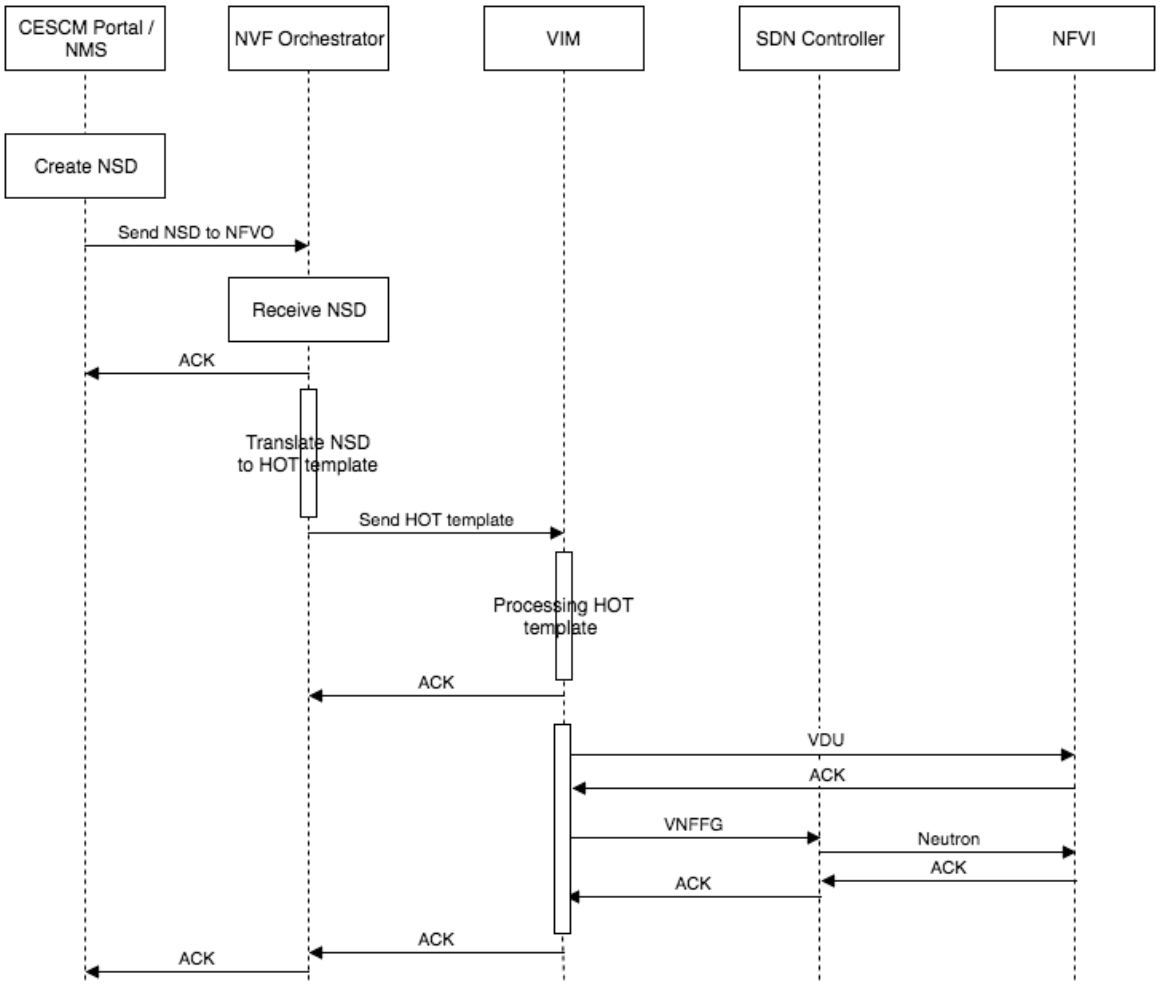

Figure 14: Descriptor workflow

5.4 Descriptor schemas

The descriptor schemas, similar to the traditional XML schemas, are used to both define the structure of the descriptors and validate them. Schemas provide the parameters to be included, defining their type (e.g., string, integer, list, key-value pair…) and the hierarchy in which they are expected to be found.

In order to verify the integrity of the descriptors coming into the CESCM and to avoid errors in the system, they need to be validated by the NFVO against the corresponding schema.

A first version of the SESAME schemas is provided in Annexes C, D and E, for the VNFD, PNFD and NSD, respectively. They are presented in JSON format and contain the parameters that are expected to be required by the CESCM and VIM modules.

Notice that, although the presented schemas will be used as the basis in development work, they are subject to future changes driven by implementation needs. Because of that, an up-to-date version of the schemas will be available at the Github repository of the project[37].

[37] https://github.com/H2020-SESAME

A basic NSD using TOSCA+yml and T-NOVA+json are included in Annex A and B, respectively, as examples of a possible implementation of the descriptor schemas. The actual descriptors will be developed as part of the implementation work and will also be available at the repository.

6 Conclusions

This deliverable reports the result of the analysis of Task 5.1, and related to the definition of the SESAME Cloud Enable Small Cell abstraction model. In the first step of the analysis, a state of the art survey has been carried out, aimed at analysing the most common models that could be used in SESAME. The ETSI information model proposed in the context of Network Function Virtualisation has been considered, and the most relevant aspects for the CESC architecture proposed by SESAME have been reported and briefly discussed. Common data modelling languages are then analysed: TOSCA, YANG, the FP7 T-Nova project models and OpenStack Heat Orchestration Templates are discussed, and some meaningful examples are reported. Finally, based on the comparison of the available solutions, the SESAME descriptors are outlined, and guidelines for their use are provided.

The major result obtained in this activity is that the available models given in the literature can be used in SESAME, without any significant modification or restriction. This renders the SESAME project open to include the results achieved in other research or open source initiatives.

The re-use in the project of the T-Nova orchestration framework as a starting point for the SESAME orchestrator, and of OpenStack for the VIM, suggests to use two different types of descriptors as internal data structures in SESAME. T-Nova descriptors can be used as the basis to create the SESAME descriptors used by the Orchestrator. Such SESAME descriptors must then be translated into Heat Orchestration Template descriptors to be forwarded to the VIM.

A first version of the descriptors used in SESAME and in particular by the Orchestrator, is reported in the Annexes of this document, to provide the first basic versions to be used by SESAME implementers. Though, a reference version, continuously updated, will be always available in the project repositories, to take into account all the modifications that will be necessarily introduced during the development phase of the project.

7 References

[1] "ETSI, "Network Functions Virtualisation (NFV); Management and Orchestration", V1.1.1, December 2014.".

[2] "http://docs.oasis-open.org/tosca/TOSCA/v1.0".

[3] A. W. S. S. B. Hedstrom, "Protocol Efficiencies of NETCONF versus SNMP for Configuration Management Functions," http://morse.colorado.edu/~tlen5710/11s/11NETCONFvsSNMP.pdf, 2011.

[4] "T-Nova project, http://www.t-nova.eu/".

[5] "https://github.com/openstack/heat-translator".

[6] "D2.3, "Specification of the CESC component". SESAMEe project".

[7] "OpenStack Orchestration API (http://developer.openstack.org/api-ref-orchestration-v1.html)".

[8] "OpenStack REST API (http://developer.openstack.org/api-ref.html)".

[9] "https://github.com/openstack/heat-templates".

[10] D4.1 Light DC architecture design. SESAME project.

[11] "https://wiki.openstack.org/wiki/Senlin".

[12] "https://www.oasis-open.org".

[13] "https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=tosca".

[14] "http://getcloudify.org".

[15] "http://alien4cloud.github.io/index.html".

[16] "https://tools.ietf.org/html/rfc6020".

[17] "TM Forum, "TM Forum Website", http://tmforum.org".

[18] D3.41 - Service Provisioning, Management and Monitoring. T-NOVA project.

[19] D5.31 - Network Functions Implementation and testing. T-NOVA project.

[20] "D6.1 – Service Description Framework. T-Nova project".

[21] https://www.openstack.org/summit/austin-2016/summit-schedule/events/7380.

[22] C. Chapper, "Deploying Virtual Network Functions: the complementary role of TOSCA and NETCONF/YANG," http://tail-f.com/wordpress/wp-content/uploads/2015/02/HR-Cisco-ALU-TOSCA-YANG-WP-2-17-15.pdf, February, 2015.

Annex A: TOSCA-NFV NSD for example SESAME NS

Example NSD in TOSCA-NFV language for the example SESAME NS provided in D6.1 section 5.2.

```
tosca_definitions_version: tosca_simple_profile_for_nfv_1_0_0
tosca_default_namespace: # Optional. default namespace (schema, types version)
description: example for a NSD.
metadata:
    ID:                 # ID of this Network Service Descriptor
    vendor:             # Provider or vendor of the Network Service
    version:            # Version of the Network Service Descriptor
imports:
     - tosca_base_type_definition.yaml
     # list of import statements for importing other definitions files
topology_template:
     inputs:
        flavour ID:

      VNF1:
        type: tosca.nodes.nfv.VNF.VNF1
        properties:
            Scaling_methodology:
            Flavour_ID:
            Threshold:
            Auto-scale policy value:
            Constraints:
        requirements:
            virtualLink1: VL1 # the subst mappings in VNF1 has virtualLink1: [CP11, virtualLink]
            virtualLink2: VL3 # the subst mappings in VNF1 has virtualLink2: [CP12, virtualLink]
            virtualLink3: VL5 # the subst mappings in VNF1 has virtualLink3: [CP13, virtualLink]
        capabilities:
            forwarder1 # the subsititution mappings in VNF1 has forwarder1: [CP11, forwarder]
            forwarder2 # the subsititution mappings in VNF1 has forwarder2: [CP12, forwarder]
            forwarder3 # the subsititution mappings in VNF1 has forwarder3: [CP13, forwarder]

    VNF2:
        type: tosca.nodes.nfv.VNF.VNF2
        properties:
            Scaling_methodology:
            Flavour_ID:
            Threshold:
            Auto-scale policy value:
            Constraints:
        requirements:
            virtualLink1: VL2 # the subst mappings in VNF2 has virtualLink1: [CP21, virtualLink]
            virtualLink2: VL3 # the subst mappings in VNF2 has virtualLink2: [CP22, virtualLink]
            virtualLink3: VL5 # the subst mappings in VNF2 has virtualLink3: [CP23, virtualLink]
        capabilities:
            forwarder1 # the subsititution mappings in VNF2 has forwarder1: [CP21, forwarder]
            forwarder2 # the subsititution mappings in VNF2 has forwarder2: [CP22, forwarder]
            forwarder3 # the subsititution mappings in VNF2 has forwarder3: [CP23, forwarder]

    VNF3:
        type: tosca.nodes.nfv.VNF.VNF3
        properties:
            Scaling_methodology:
            Flavour_ID:
```

```
            Threshold:
            Auto-scale policy value:
            Constraints:
        requirements:
            virtualLink1: VL3 # the subst mappings in VNF3 has virtualLink1: [CP32, virtualLink]
            virtualLink2: VL4 # the subst mappings in VNF3 has virtualLink2: [CP31, virtualLink]
        capabilities:
            forwarder1 # the subst mappings in VNF3 has forwarder1: [CP32, forwarder]
            forwarder2 # the subst mappings in VNF3 has forwarder2: [CP31, forwarder]

    VNF4:
        type: tosca.nodes.nfv.VNF.VNF4
        properties:
            Scaling_methodology:
            Flavour_ID:
            Threshold:
            Auto-scale policy value:
            Constraints:
        requirements:
            virtualLink1: VL5 # the subst mappings in VNF4 has virtualLink1: [CP41, virtualLink]
        capabilities:
            forwarder1 # the subsititution mappings in VNF4 has forwarder1: [CP41, forwarder]

    VNF5:
        type: tosca.nodes.nfv.VNF.VNF5
        properties:
            Scaling_methodology:
            Flavour_ID:
            Threshold:
            Auto-scale policy value:
            Constraints:
        requirements:
            virtualLink1: VL5 # the subst mappings in VNF5 has virtualLink1: [CP51, virtualLink]
        capabilities:
            forwarder1 # the subst mappings in VNF5 has forwarder1: [CP51, forwarder]


    CP01 #endpoints of NS
        type: tosca.nodes.nfv.CP
        properties:
            type:
        requirements:
            virtualLink: VL1

    CP02 #endpoints of NS
        type: tosca.nodes.nfv.CP
        properties:
            type:
        requirements:
            virtualLink: VL2

    CP03 #endpoints of NS
        type: tosca.nodes.nfv.CP
        properties:
            type:
        requirements:
            virtualLink: VL4
```

```
    VL1
        type: tosca.nodes.nfv.VL.Eline
        properties:
            # omitted here for brevity
        capabilities:
            -virtual_linkable
            occurrences: 2

    VL2
        type: tosca.nodes.nfv.VL.ELine
        properties:
            # omitted here for brevity
        capabilities:
            -virtual_linkable
            occurrences: 5

    VL3
        type: tosca.nodes.nfv.VL.ELAN
        properties:
            # omitted here for brevity
        capabilities:
            -virtual_linkable
            occurrences: 2

    VL4
        type: tosca.nodes.nfv.VL.Eline
        properties:
            # omitted here for brevity
        capabilities:
            -virtual_linkable
            occurrences: 2

    VL5
        type: tosca.nodes.nfv.VL.ELAN
        properties:
            # omitted here for brevity
        capabilities:
            -virtual_linkable
            occurrences: 2
    Forwarding path1:
        type: tosca.nodes.nfv.FP
        description: the path (CP01_CP11_CP12_CP32_CP31_CP03)
        properties:
            policy:
        requirements:
            -forwarder: CP01
            -forwarder: VNF1
                capability: forwarder1 #CP11
            -forwarder: VNF1
                capability: forwarder2 #CP12
            -forwarder: VNF3
                capability: forwarder1 #CP32
            -forwarder: VNF3
                capability: forwarder2 #CP31
            -forwarder: CP03

    Forwarding path2:
        type: tosca.nodes.nfv.FP
```

```
        description: the path (CP02_CP21_CP22_CP32_CP31_CP03)
        properties:
            policy:
        requirements:
            -forwarder: CP02
            -forwarder: VNF2
                capability: forwarder1 #CP21
            -forwarder: VNF2
                capability: forwarder2 #CP22
            -forwarder: VNF3
                capability: forwarder1 #CP32
            -forwarder: VNF3
                capability: forwarder2 #CP31
            -forwarder: CP03

    Forwarding path3:
        type: tosca.nodes.nfv.FP
        description: the path (CP01_CP11_CP13_CP41_CP13_CP12_CP32_CP31_CP03)
        properties:
            policy:
        requirements:
            -forwarder: CP01
            -forwarder: VNF1
                capability: forwarder1 #CP11
            -forwarder: VNF1
                capability: forwarder3 #CP13
            -forwarder: VNF4
                capability: forwarder1 #CP41
            -forwarder: VNF1
                capability: forwarder3 #CP13
            -forwarder: VNF1
                capability: forwarder2 #CP12
            -forwarder: VNF3
                capability: forwarder1 #CP32
            -forwarder: VNF3
                capability: forwarder2 #CP31
            -forwarder: CP03

    Forwarding path4:
        type: tosca.nodes.nfv.FP
        description: the path (CP02_CP21_CP23_CP41_CP23_CP22_CP32_CP31_CP03)
        properties:
            policy:
        requirements:
             -forwarder: CP02
             -forwarder: VNF2
                 capability: forwarder1 #CP21
             -forwarder: VNF2
                capability: forwarder3 #CP23
            -forwarder: VNF4
                capability: forwarder1 #CP41
            -forwarder: VNF2
                capability: forwarder3 #CP23
            -forwarder: VNF2
                capability: forwarder2 #CP22
            -forwarder: VNF3
                capability: forwarder1 #CP32
            -forwarder: VNF3
```

```
            capability: forwarder2 #CP31
        -forwarder: CP03

Forwarding path5:
    type: tosca.nodes.nfv.FP
    description: the path (CP01_CP11_CP13_CP51_CP13_CP11_CP01)
    properties:
        policy:
    requirements:
         -forwarder: CP01
        -forwarder: VNF1
            capability: forwarder1 #CP11
        -forwarder: VNF1
            capability: forwarder3 #CP13
        -forwarder: VNF5
            capability: forwarder1 #CP51
        -forwarder: VNF1
            capability: forwarder3 #CP13
        -forwarder: VNF1
            capability: forwarder1 #CP11
        -forwarder: CP01

Forwarding path6:
    type: tosca.nodes.nfv.FP
    description: the path (CP01_CP11_CP13_CP51_CP13_CP12_CP32_CP31_CP03)
    properties:
        policy:
    requirements:
        -forwarder: CP01
        -forwarder: VNF1
            capability: forwarder1 #CP11
        -forwarder: VNF1
            capability: forwarder3 #CP13
        -forwarder: VNF5
            capability: forwarder1 #CP51
        -forwarder: VNF1
            capability: forwarder3 #CP13
        -forwarder: VNF1
            capability: forwarder2 #CP12
        -forwarder: VNF3
            capability: forwarder1 #CP32
        -forwarder: VNF3
            capability: forwarder2 #CP31
        -forwarder: CP03

Forwarding path7:
    type: tosca.nodes.nfv.FP
    description: the path (CP02_CP21_CP23_CP51_CP23_CP21_CP02)
    properties:
        policy:
    requirements:
        -forwarder: CP02
        -forwarder: VNF2
            capability: forwarder1 #CP21
        -forwarder: VNF2
            capability: forwarder3 #CP23
        -forwarder: VNF5
            capability: forwarder1 #CP51
```

```
            -forwarder: VNF2
                capability: forwarder3 #CP23
            -forwarder: VNF2
                capability: forwarder1 #CP21
            -forwarder: CP02

    Forwarding path8:
        type: tosca.nodes.nfv.FP
        description: the path (CP02_CP21_CP23_CP51_CP23_CP22_CP32_CP31_CP03)
        properties:
            policy:
        requirements:
            -forwarder: CP02
            -forwarder: VNF2
                capability: forwarder1 #CP21
            -forwarder: VNF2
                capability: forwarder3 #CP23
            -forwarder: VNF5
                capability: forwarder1 #CP51
            -forwarder: VNF2
                capability: forwarder3 #CP23
            -forwarder: VNF2
                capability: forwarder2 #CP22
            -forwarder: VNF3
                capability: forwarder1 #CP32
            -forwarder: VNF3
                capability: forwarder2 #CP31
            -forwarder: CP03

    Groups:
        VNFFG1:
            type: tosca.groups.nfv.vnffg
            description: C-Plane
            properties:
                vendor:
                version:
                vl: [VL1,VL2,VL3,VL4]
                vnf: [VNF1,VNF2,VNF3]
            targets: [Forwarding path1, Forwarding path2]

        VNFFG2:
            type: tosca.groups.nfv.vnffg
            description: U-Plane External Services
            properties:
                vendor:
                version:
                vl: [VL1,VL2,VL3,VL4]
                vnf: [VNF1,VNF2,VNF3]
            targets: [Forwarding path1, Forwarding path2]

        VNFFG3:
            type: tosca.groups.nfv.vnffg
            description: U-Plane External Services Edge Content Mod
            properties:
                vendor:
                version:
                vl: [VL1,VL2,VL3,VL4,VL5]
                vnf: [VNF1,VNF2,VNF3,VNF4]
```

```
            targets: [Forwarding path3, Forwarding path4]

        VNFFG4:
            type: tosca.groups.nfv.vnffg
            description: U-Plane External Services Transparent Caching
            properties:
                 vendor:
                 version:
                 vl: [VL1,VL2,VL3,VL4,VL5]
                 vnf: [VNF1,VNF2,VNF3,VNF5]
            targets: [Forwarding path5, Forwarding path6, Forwarding path7, Forwarding path8]

        VNFFG5:
            type: tosca.groups.nfv.vnffg
            description: U-Plane LocalServices
            properties:
                vendor:
                version:
                 vl: [VL1,VL2,VL5]
                vnf: [VNF1,VNF2,VNF4,VNF5]
            targets: [Forwarding path5, Forwarding path7]
```

Annex B: T-NOVA NSD for example SESAME NS

Example NSD in T-NOVA (ETSI) language for the example SESAME NS provided in D6.1 section 5.2.

```
{
  "nsd": {
    "id": "",           //ID of this Network Service Descriptor
    "name": "",         //Name of the Network Service
    "vendor": "",       //Provider or vendor of the Network Service
    "version": "",      //Version of the Network Service Descriptor
    "provider_id": "",  //Id of the vendor of the Network Service
    "description": "Example for a NSD",
    "vnfds": [
        "1",
        "2",
        "3",
        "4",
        "5"
    ],
    "vnffgd": {
      "vnffgs": [
        {
          "vnffg_id": "vnffg1",
          "number_of_endpoints": 3,
          "number_of_virtual_links": 4,
          "dependent_virtual_links": [
            "vld1",
            "vld2",
            "vld3",
            "vld4"
          ],
          "network_forwarding_path": [
            {
              "nfp_id": "nfp1",
              "graph": [
                "vld1",
                "vld3",
                "vld4"
              ],
              "connection_points": [
                "CP01",
                "VNF#1:CP11",
                "VNF#1:CP12",
                "VNF#3:CP32",
                "VNF#3:CP31",
                "CP03"
              ],
              "constituent_vnfs": [
                {
                  "vnf_ref_id": "1",
                  "vnf_flavor_key_ref": ""
                },
                {
                  "vnf_ref_id": "3",
                  "vnf_flavor_key_ref": ""
                }
```

```
            ]
          },
          {
            "nfp_id": "nfp2",
            "graph": [
              "vld2",
              "vld3",
              "vld4"
            ],
            "connection_points": [
              "CP02",
              "VNF#2:CP21",
              "VNF#2:CP22",
              "VNF#3:CP32",
              "VNF#3:CP31",
              "CP03"
            ],
            "constituent_vnfs": [
              {
                "vnf_ref_id": "2",
                "vnf_flavor_key_ref": ""
              },
              {
                "vnf_ref_id": "3",
                "vnf_flavor_key_ref": ""
              }
            ]
          }
        ]
      },
      {
        "vnffg_id": "vnffg2",
        "number_of_endpoints": 3,
        "number_of_virtual_links": 4,
        "dependent_virtual_links": [
          "vld1",
          "vld2",
          "vld3",
          "vld4"
        ],
        "network_forwarding_path": [
          {
            "nfp_id": "nfp1",
            "graph": [
              "vld1",
              "vld3",
              "vld4"
            ],
            "connection_points": [
              "CP01",
              "VNF#1:CP11",
              "VNF#1:CP12",
              "VNF#3:CP32",
              "VNF#3:CP31",
              "CP03"
            ],
            "constituent_vnfs": [
              {
```

```
                "vnf_ref_id": "1",
                "vnf_flavor_key_ref": ""
              },
              {
                "vnf_ref_id": "3",
                "vnf_flavor_key_ref": ""
              }
            ]
          },
          {
            "nfp_id": "nfp2",
            "graph": [
              "vld2",
              "vld3",
              "vld4"
            ],
            "connection_points": [
              "CP02",
              "VNF#2:CP21",
              "VNF#2:CP22",
              "VNF#3:CP32",
              "VNF#3:CP31",
              "CP03"
            ],
            "constituent_vnfs": [
              {
                "vnf_ref_id": "2",
                "vnf_flavor_key_ref": ""
              },
              {
                "vnf_ref_id": "3",
                "vnf_flavor_key_ref": ""
              }
            ]
          }
        ]
      },
            {
        "vnffg_id": "vnffg3",
        "number_of_endpoints": 3,
        "number_of_virtual_links": 5,
        "dependent_virtual_links": [
          "vld1",
          "vld2",
          "vld3",
          "vld4",
          "vld5"
        ],
        "network_forwarding_path": [
          {
            "nfp_id": "nfp3",
            "graph": [
              "vld1",
              "vld5",
              "vld3",
              "vld4"
            ],
            "connection_points": [
```

```
            "CP01",
            "VNF#1:CP11",
            "VNF#1:CP13",
            "VNF#4:CP41",
            "VNF#1:CP13",
            "VNF#1:CP12",
            "VNF#3:CP32",
            "VNF#3:CP31",
            "CP03"
          ],
          "constituent_vnfs": [
            {
              "vnf_ref_id": "1",
              "vnf_flavor_key_ref": ""
            },
            {
              "vnf_ref_id": "4",
              "vnf_flavor_key_ref": ""
            },
            {
              "vnf_ref_id": "3",
              "vnf_flavor_key_ref": ""
            }
          ]
        },
        {
          "nfp_id": "nfp4",
          "graph": [
            "vld2",
            "vld3",
            "vld5",
            "vld4"
          ],
          "connection_points": [
            "CP02",
            "VNF#2:CP21",
            "VNF#2:CP23",
            "VNF#4:CP41",
            "VNF#2:CP23",
            "VNF#2:CP22",
            "VNF#3:CP32",
            "VNF#3:CP31",
            "CP03"
          ],
          "constituent_vnfs": [
            {
              "vnf_ref_id": "2",
              "vnf_flavor_key_ref": ""
            },
            {
              "vnf_ref_id": "4",
              "vnf_flavor_key_ref": ""
            },
            {
              "vnf_ref_id": "3",
              "vnf_flavor_key_ref": ""
            }
          ]
```

```
            }
          ]
        },
        {
          "vnffg_id": "vnffg4",
          "number_of_endpoints": 3,
          "number_of_virtual_links": 5,
          "dependent_virtual_links": [
            "vld1",
            "vld2",
            "vld3",
            "vld4",
            "vld5"
          ],
          "network_forwarding_path": [
            {
              "nfp_id": "nfp5",
              "graph": [
                "vld1",
                "vld5"
              ],
              "connection_points": [
                "CP01",
                "VNF#1:CP11",
                "VNF#1:CP13",
                "VNF#5:CP51",
                "VNF#1:CP13",
                "VNF#1:CP11",
                "CP01"
              ],
              "constituent_vnfs": [
                {
                  "vnf_ref_id": "1",
                  "vnf_flavor_key_ref": ""
                },
                {
                  "vnf_ref_id": "5",
                  "vnf_flavor_key_ref": ""
                }
              ]
            },
            {
              "nfp_id": "nfp6",
              "graph": [
                "vld1",
                "vld5",
                "vld3",
                "vld4"
              ],
              "connection_points": [
                "CP01",
                "VNF#1:CP11",
                "VNF#1:CP13",
                "VNF#5:CP51",
                "VNF#1:CP13",
                "VNF#1:CP12",
                "VNF#3:CP32",
                "VNF#3:CP31",
```

```
            "CP03"
          ],
          "constituent_vnfs": [
            {
              "vnf_ref_id": "1",
              "vnf_flavor_key_ref": ""
            },
            {
              "vnf_ref_id": "5",
              "vnf_flavor_key_ref": ""
            },
            {
              "vnf_ref_id": "3",
              "vnf_flavor_key_ref": ""
            }
          ]
        },
        {
          "nfp_id": "nfp7",
          "graph": [
            "vld2",
            "vld5"
          ],
          "connection_points": [
            "CP02",
            "VNF#1:CP21",
            "VNF#1:CP23",
            "VNF#5:CP51",
            "VNF#1:CP23",
            "VNF#1:CP21",
            "CP02"
          ],
          "constituent_vnfs": [
            {
              "vnf_ref_id": "2",
              "vnf_flavor_key_ref": ""
            },
            {
              "vnf_ref_id": "5",
              "vnf_flavor_key_ref": ""
            }
          ]
        },
        {
          "nfp_id": "nfp8",
          "graph": [
            "vld2",
            "vld5",
            "vld3",
            "vld4"
          ],
          "connection_points": [
            "CP02",
            "VNF#2:CP21",
            "VNF#2:CP23",
            "VNF#5:CP51",
            "VNF#2:CP23",
            "VNF#2:CP22",
```

```
              VNF#3:CP32",
              "VNF#3:CP31",
              "CP03"
            ],
            "constituent_vnfs": [
              {
                "vnf_ref_id": "2",
                "vnf_flavor_key_ref": ""
              },
              {
                "vnf_ref_id": "5",
                "vnf_flavor_key_ref": ""
              },
              {
                "vnf_ref_id": "3",
                "vnf_flavor_key_ref": ""
              }
            ]
          }
        ]
      },
      {
        "vnffg_id": "vnffg5",
        "number_of_endpoints": 2,
        "number_of_virtual_links": 3,
        "dependent_virtual_links": [
          "vld1",
          "vld2",
          "vld5"
        ],
        "network_forwarding_path": [
          {
            "nfp_id": "nfp5",
            "graph": [
              "vld1",
              "vld5"
            ],
            "connection_points": [
              "CP01",
              "VNF#1:CP11",
              "VNF#1:CP13",
              "VNF#5:CP51",
              "VNF#1:CP13",
              "VNF#1:CP11",
              "CP01"
            ],
            "constituent_vnfs": [
              {
                "vnf_ref_id": "1",
                "vnf_flavor_key_ref": ""
              },
              {
                "vnf_ref_id": "5",
                "vnf_flavor_key_ref": ""
              }
            ]
          },
          {
```

```
            "nfp_id": "nfp7",
            "graph": [
              "vld2",
              "vld5"
            ],
            "connection_points": [
              "CP02",
              "VNF#1:CP21",
              "VNF#1:CP23",
              "VNF#5:CP51",
              "VNF#1:CP23",
              "VNF#1:CP21",
              "CP02"
            ],
            "constituent_vnfs": [
              {
                "vnf_ref_id": "2",
                "vnf_flavor_key_ref": ""
              },
              {
                "vnf_ref_id": "5",
                "vnf_flavor_key_ref": ""
              }
            ]
          }
        ]
      }
    ]
  },
  "lifecycle_events": {
    "start": [
      {
        "vnf_id": "4",
        "vnf_event": "start",
        "event_rel_id": "rel_le0",
        "event_id": "le0",
        "sla_ref_id": "sla0"
      },
      {
        "vnf_id": "5",
        "vnf_event": "start",
        "event_rel_id": "rel_le0",
        "event_id": "le0",
        "sla_ref_id": "sla0"
      }
    ],
    "stop": [
        {
        "vnf_id": "4",
        "vnf_event": "stop",
        "event_rel_id": "rel_le1",
        "event_id": "le1",
        "sla_ref_id": "sla0"
      },
      {
        "vnf_id": "5",
        "vnf_event": "stop",
        "event_rel_id": "rel_le1",
```

```
        "event_id": "le1",
        "sla_ref_id": "sla0"
      }
        ],
    "scale_out": []
  },
  "monitoring_parameters": [
    {
      "desc": "Availability",
      "metric": "availability",
      "unit": "%"
    },
    {
      "desc": "End-to-End Bandwidth",
      "metric": "end-to-end bandwidth",
      "unit": "Mbps"
    }
  ],
  "vld": {
    "number_of_endpoints": 0,
    "virtual_links": [
      {
        "vld_id": "vld1",
        "alias": "VL1",
        "root_requirements": "Unlimited",
        "leaf_requirement": "Unlimited",
        "qos": {
          "average": "",
          "peak": "",
          "burst": ""
        },
        "connections": [
          "CP01",
          "VNF#1:CP11"
        ],
        "connectivity_type": "E-LAN",
        "external_access": true,
        "merge": false,
        "sla_ref_id": "sla0"
      },
      {
        "vld_id": "vld2",
        "alias": "VL2",
        "root_requirements": "Unlimeted",
        "leaf_requirement": "Unlimeted",
        "qos": {
          "average": "",
          "peak": "",
          "burst": ""
        },
        "connections": [
          "CP02",
          "VNF#2:CP21"
        ],
        "connectivity_type": "E-LINE",
        "external_access": true,
        "merge": false,
        "sla_ref_id": "sla0"
```

```
      },
      {
        "vld_id": "vld3",
        "alias": "VL3",
        "root_requirements": "Unlimeted",
        "leaf_requirement": "Unlimeted",
        "qos": {
          "average": "",
          "peak": "",
          "burst": ""
        },
        "connections": [
          "VNF#1:CP12",
          "VNF#2:CP22",
          "VNF#2:CP23",
          "VNF#3:CP32"
        ],
        "connectivity_type": "E-LINE",
        "external_access": false,
        "merge": true,
        "sla_ref_id": "sla0"
      },
      {
        "vld_id": "vld4",
        "alias": "VL4",
        "root_requirements": "Unlimeted",
        "leaf_requirement": "Unlimeted",
        "qos": {
          "average": "",
          "peak": "",
          "burst": ""
        },
        "connections": [
          "CP03",
          "VNF#3:CP31"
        ],
        "connectivity_type": "E-LINE",
        "external_access": true,
        "merge": false,
        "sla_ref_id": "sla0"
      },
      {
        "vld_id": "vld5",
        "alias": "VL5",
        "root_requirements": "Unlimeted",
        "leaf_requirement": "Unlimeted",
        "qos": {
          "average": "",
          "peak": "",
          "burst": ""
        },
        "connections": [
          "VNF#1:CP13",
          "VNF#2:CP23",
          "VNF#4:CP41",
          "VNF#5:CP51"
        ],
        "connectivity_type": "E-LINE",
```

```
        "external_access": false,
        "merge": true,
        "sla_ref_id": "sla0"
      }
    ]
  },
  "sla": [
    {
      "id": "sla0",
      "assurance_parameters": [
        {
          "formula": "MAX(VNF:5.httpnum)",
          "id": "httpnum",
          "name": "httpnum",
          "penalty": {
            "type": "Discount",
            "unit": "INT",
            "validity": "P1D",
            "value": 10
          },
          "unit": "INT",
          "value": "GT(100)",
          "violations": [
            {
              "breaches_count": 2,
              "interval": 360
            }
          ]
        }
      ],
      "billing": {
        "model": "PAYG",
        "price": {
          "price_per_period": 30,
          "setup": 10,
          "unit": "EUR"
        }
      },
      "constituent_vnf": [
        {
          "number_of_instances": 1,
          "redundancy_model": "Active",
          "vnf_flavour_id_reference": "gold",
          "vnf_reference": "1"
        },
        {
          "number_of_instances": 1,
          "redundancy_model": "Active",
          "vnf_flavour_id_reference": "gold",
          "vnf_reference": "2"
        },
        {
          "number_of_instances": 1,
          "redundancy_model": "Active",
          "vnf_flavour_id_reference": "gold",
          "vnf_reference": "3"
        },
        {
```

```
            "number_of_instances": 1,
            "redundancy_model": "Active",
            "vnf_flavour_id_reference": "gold",
            "vnf_reference": "4"
          },
          {
            "number_of_instances": 1,
            "redundancy_model": "Active",
            "vnf_flavour_id_reference": "gold",
            "vnf_reference": "5"
          }
        ],
        "sla_key": "Basic"
      }
    ],
    "auto_scale_policy": {
      "criteria": []
    },
    "connection_points": [
                {
                        "id": "CP01",
                        "type":  "bridge",
                        "interface": 1
                },
                {
                        "id": "CP02",
                        "type":  "bridge",
                        "interface": 1
                },
                {
                        "id": "CP03",
                        "type":  "bridge",
                        "interface": 1
                }
        ],
    "vnf_dependency": []

  }
}
```

Annex C: SESAME VNFD Schema

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "id": "",
  "type": "object",
  "properties": {
    "release": {
      "id": "/vnfd/release",
      "type": "string"
    },
       "id": {
      "id": "/vnfd/id",
      "type": "integer"
    },
       "vendor": {
      "id": "/vnfd/vendor",
      "type": "string"
    },
       "descriptor_version": {
      "id": "/vnfd/descriptor_version",
      "type": "string"
    },
        "version": {
      "id": "/vnfd/version",
      "type": "string"
    },
       "name": {
      "id": "/vnfd/name",
      "type": "string"
    },
       "created_at": {
      "id": "/vnfd/created_at",
      "type": "string"
    },
    "modified_at": {
      "id": "/vnfd/modified_at",
      "type": "string"
    },
       "manifest_file_md5": {
          "id": "/vnfd/manifest_file_md5",
          "type": "string"
        },
    "type": {
      "id": "/vnfd/type",
      "type": "string"
    },
    "description": {
      "id": "/vnfd/description",
      "type": "string"
    },
    "vdu": {
      "id": "/vnfd/vdu",
      "type": "array",
      "items": {
        "id": "/vnfd/vdu/1",
```

```
        "type": "object",
        "properties": {
          "resource_requirements": {
            "id": "/vnfd/vdu/1/resource_requirements",
            "type": "object",
            "properties": {
              "network_interface_bandwidth_unit": {
                "id":
"/vnfd/vdu/1/resource_requirements/network_interface_bandwidth_unit",
                "type": "string"
              },
              "hypervisor_parameters": {
                "id": "/vnfd/vdu/1/resource_requirements/hypervisor_parameters",
                "type": "object",
                "properties": {
                  "version": {
                    "id":
"/vnfd/vdu/1/resource_requirements/hypervisor_parameters/version",
                    "type": "string"
                  },
                  "type": {
                    "id":
"/vnfd/vdu/1/resource_requirements/hypervisor_parameters/type",
                    "type": "string"
                  }
                },
                "additionalProperties": false
              },
              "memory_unit": {
                "id": "/vnfd/vdu/1/resource_requirements/memory_unit",
                "type": "string"
              },
              "network_interface_card_capabilities": {
                "id":
"/vnfd/vdu/1/resource_requirements/network_interface_card_capabilities",
                "type": "object",
                "properties": {
                  "SR-IOV": {
                    "id":
"/vnfd/vdu/1/resource_requirements/network_interface_card_capabilities/SR-IOV",
                    "type": "boolean"
                  },
                  "mirroring": {
                    "id":
"/vnfd/vdu/1/resource_requirements/network_interface_card_capabilities/mirroring",
                    "type": "boolean"
                  }
                },
                "additionalProperties": false
              },
              "storage": {
                "id": "/vnfd/vdu/1/resource_requirements/storage",
                "type": "object",
                "properties": {
                  "size_unit": {
                    "id": "/vnfd/vdu/1/resource_requirements/storage/size_unit",
                    "type": "string"
                  },
```

```
                  "persistence": {
                    "id": "/vnfd/vdu/1/resource_requirements/storage/persistence",
                    "type": "boolean"
                  },
                  "size": {
                    "id": "/vnfd/vdu/1/resource_requirements/storage/size",
                    "type": "integer"
                  }
                },
                "additionalProperties": false
              },
              "network_interface_bandwidth": {
                "id": "/vnfd/vdu/1/resource_requirements/network_interface_bandwidth",
                "type": "string"
              },
              "platform_pcie_parameters": {
                "id": "/vnfd/vdu/1/resource_requirements/platform_pcie_parameters",
                "type": "object",
                "properties": {
                  "SR-IOV": {
                    "id":
"/vnfd/vdu/1/resource_requirements/platform_pcie_parameters/SR-IOV",
                    "type": "boolean"
                  },
                  "device_pass_through": {
                    "id":
"/vnfd/vdu/1/resource_requirements/platform_pcie_parameters/device_pass_through",
                    "type": "boolean"
                  }
                },
                "additionalProperties": false
              },
              "vcpus": {
                "id": "/vnfd/vdu/1/resource_requirements/vcpus",
                "type": "integer"
              },
              "vswitch_capabilities": {
                "id": "/vnfd/vdu/1/resource_requirements/vswitch_capabilities",
                "type": "object",
                "properties": {
                  "version": {
                    "id":
"/vnfd/vdu/1/resource_requirements/vswitch_capabilities/version",
                    "type": "string"
                  },
                  "type": {
                    "id":
"/vnfd/vdu/1/resource_requirements/vswitch_capabilities/type",
                    "type": "string"
                  },
                  "overlay_tunnel": {
                    "id":
"/vnfd/vdu/1/resource_requirements/vswitch_capabilities/overlay_tunnel",
                    "type": "string"
                  }
                },
                "additionalProperties": false
              },
```

```
              "data_processing_acceleration_library": {
                "id":
"/vnfd/vdu/1/resource_requirements/data_processing_acceleration_library",
                "type": "string"
              },
              "memory": {
                "id": "/vnfd/vdu/1/resource_requirements/memory",
                "type": "integer"
              },
              "memory_parameters": {
                "id": "/vnfd/vdu/1/resource_requirements/memory_parameters",
                "type": "object",
                "properties": {
                  "large_pages_required": {
                    "id":
"/vnfd/vdu/1/resource_requirements/memory_parameters/large_pages_required",
                    "type": "boolean"
                  },
                  "numa_allocation_policy": {
                    "id":
"/vnfd/vdu/1/resource_requirements/memory_parameters/numa_allocation_policy",
                    "type": "string"
                  }
                },
                "additionalProperties": false
              },
              "cpu_support_accelerator": {
                "id": "/vnfd/vdu/1/resource_requirements/cpu_support_accelerator",
                "type": "string"
              }
            },
            "additionalProperties": false
          },
          "bootstrap_script": {
            "id": "/vnfd/vdu/1/bootstrap_script",
            "type": "string"
          },
          "vm_image": {
            "id": "/vnfd/vdu/1/vm_image",
            "type": "string"
          },
          "vm_image_format": {
            "id": "/vnfd/vdu/1/vm_image_format",
            "type": "string"
          },
          "networking_resources": {
            "id": "/vnfd/vdu/1/networking_resources",
            "type": "string"
          },
          "monitoring_parameters_specific": {
            "id": "/vnfd/vdu/1/monitoring_parameters_specific",
            "type": "array",
            "items": {
              "id": "/vnfd/vdu/1/monitoring_parameters_specific/6",
              "type": "object",
              "properties": {
                "metric": {
                  "id": "/vnfd/vdu/1/monitoring_parameters_specific/6/metric",
```

```
                "type": "string"
              },
              "desc": {
                "id": "/vnfd/vdu/1/monitoring_parameters_specific/6/desc",
                "type": "string"
              },
              "unit": {
                "id": "/vnfd/vdu/1/monitoring_parameters_specific/6/unit",
                "type": "string"
              }
            },
            "additionalProperties": false
          }
        },
        "id": {
          "id": "/vnfd/vdu/1/id",
          "type": "string"
        },
        "alias": {
          "id": "/vnfd/vdu/1/alias",
          "type": "string"
        },
        "controller": {
          "id": "/vnfd/vdu/1/controller",
          "type": "boolean"
        },
        "connection_points": {
          "id": "/vnfd/vdu/1/connection_points",
          "type": "array",
          "items": {
            "id": "/vnfd/vdu/1/connection_points/2",
            "type": "object",
            "properties": {
              "vlink_ref": {
                "id": "/vnfd/vdu/1/connection_points/2/vlink_ref",
                "type": "string"
              },
              "id": {
                "id": "/vnfd/vdu/1/connection_points/2/id",
                "type": "string"
              }
            },
            "additionalProperties": false
          }
        },
        "monitoring_parameters": {
          "id": "/vnfd/vdu/1/monitoring_parameters",
          "type": "array",
          "items": {
            "id": "/vnfd/vdu/1/monitoring_parameters/8",
            "type": "object",
            "properties": {
              "metric": {
                "id": "/vnfd/vdu/1/monitoring_parameters/8/metric",
                "type": "string"
              },
              "unit": {
                "id": "/vnfd/vdu/1/monitoring_parameters/8/unit",
```

```
                "type": "string"
              },
              "desc": {
                "id": "/vnfd/vdu/1/monitoring_parameters/8/desc",
                "type": "string"
              }
            },
            "additionalProperties": false
          }
        },
        "vm_image_md5": {
          "id": "/vnfd/vdu/1/vm_image_md5",
          "type": "string"
        },
        "scale_in_out": {
          "id": "/vnfd/vdu/1/scale_in_out",
          "type": "object",
          "properties": {
            "minimum": {
              "id": "/vnfd/vdu/1scale_in_out/minimum",
              "type": "integer"
            },
            "maximum": {
              "id": "/vnfd/vdu/1/scale_in_out/maximum",
              "type": "integer"
            }
          },
          "additionalProperties": false
        }
      },
      "additionalProperties": false
    }
  },

  "virtual_links": {
    "id": "/vnfd/virtual_links",
    "type": "array",
    "items": {
      "id": "/vnfd/virtual_links/3",
      "type": "object",
      "properties": {
        "leaf_requirement": {
          "id": "/vnfd/virtual_links/3/leaf_requirement",
          "type": "string"
        },
        "connectivity_type": {
          "id": "/vnfd/virtual_links/3/connectivity_type",
          "type": "string"
        },
        "vdu_reference": {
          "id": "/vnfd/virtual_links/3/vdu_reference",
          "type": "array",
          "items": {
            "id": "/vnfd/virtual_links/3/vdu_reference/0",
            "type": "string"
          }
        },
        "external_access": {
```

```
            "id": "/vnfd/virtual_links/3/external_access",
            "type": "boolean"
          },
          "connection_points_reference": {
            "id": "/vnfd/virtual_links/3/connection_points_reference",
            "type": "array",
            "items": {
              "id": "/vnfd/virtual_links/3/connection_points_reference/1",
              "type": "string"
            }
          },
          "net_segment": {
            "id": "/vnfd/virtual_links/3/net_segment",
            "type": "string"
          },
          "access": {
            "id": "/vnfd/virtual_links/3/access",
            "type": "boolean"
          },
          "alias": {
            "id": "/vnfd/virtual_links/3/alias",
            "type": "string"
          },
          "root_requirement": {
            "id": "/vnfd/virtual_links/3/root_requirement",
            "type": "string"
          },
          "dhcp": {
            "id": "/vnfd/virtual_links/3/dhcp",
            "type": "boolean"
          },
          "id": {
            "id": "/vnfd/virtual_links/3/id",
            "type": "string"
          },
          "qos": {
            "id": "/vnfd/virtual_links/3/qos",
            "type": "string"
          }
        },
        "additionalProperties": false
      }
    },
    "deployment_flavours": {
      "id": "/vnfd/deployment_flavours",
      "type": "array",
      "items": {
        "id": "/vnfd/deployment_flavours/1",
        "type": "object",
        "properties": {
          "vdu_reference": {
            "id": "/vnfd/deployment_flavours/1/vdu_reference",
            "type": "array",
            "items": {
              "id": "/vnfd/deployment_flavours/1/vdu_reference/0",
              "type": "string"
            }
          },
```

```
          "constraint": {
            "id": "/vnfd/deployment_flavours/1/constraint",
            "type": "string"
          },
          "flavour_key": {
            "id": "/vnfd/deployment_flavours/1/flavour_key",
            "type": "string"
          },
          "vlink_reference": {
            "id": "/vnfd/deployment_flavours/1/vlink_reference",
            "type": "array",
            "items": {
              "id": "/vnfd/deployment_flavours/1/vlink_reference/1",
              "type": "string"
            }
          },
          "id": {
            "id": "/vnfd/deployment_flavours/1/id",
            "type": "string"
          },
          "assurance_parameters": {
            "id": "/vnfd/deployment_flavours/1/assurance_parameters",
            "type": "array",
            "items": {
              "id": "/vnfd/deployment_flavours/1/assurance_parameters/0",
              "type": "object",
              "properties": {
                "violation": {
                  "id": "/vnfd/deployment_flavours/1/assurance_parameters/0/violation",
                  "type": "array",
                  "items": {
                    "id":
"/vnfd/deployment_flavours/1/assurance_parameters/0/violation/0",
                    "type": "object",
                    "properties": {
                      "interval": {
                        "id":
"/vnfd/deployment_flavours/1/assurance_parameters/0/violation/0/interval",
                        "type": "integer"
                      },
                      "breaches_count": {
                        "id":
"/vnfd/deployment_flavours/1/assurance_parameters/0/violation/0/breaches_count",
                        "type": "integer"
                      }
                    },
                    "additionalProperties": false
                  }
                },
                "value": {
                  "id": "/vnfd/deployment_flavours/1/assurance_parameters/0/value",
                  "type": "integer"
                },
                "penalty": {
                  "id": "/vnfd/deployment_flavours/1/assurance_parameters/0/penalty",
                  "type": "object",
                  "properties": {
                    "type": {
```

```
                      "id":
"/vnfd/deployment_flavours/1/assurance_parameters/0/penalty/type",
                      "type": "string"
                    },
                    "expression": {
                      "id":
"/vnfd/deployment_flavours/1/assurance_parameters/0/penalty/expression",
                      "type": "integer"
                    },
                    "validity": {
                      "id":
"/vnfd/deployment_flavours/1/assurance_parameters/0/penalty/validity",
                      "type": "string"
                    },
                    "unit": {
                      "id":
"/vnfd/deployment_flavours/1/assurance_parameters/0/penalty/unit",
                      "type": "string"
                    }
                  },
                  "additionalProperties": false
                },
                "formula": {
                  "id": "/vnfd/deployment_flavours/1/assurance_parameters/0/formula",
                  "type": "string"
                },
                "rel_id": {
                  "id": "/vnfd/deployment_flavours/1/assurance_parameters/0/rel_id",
                  "type": "string"
                },
                "id": {
                  "id": "/vnfd/deployment_flavours/1/assurance_parameters/0/id",
                  "type": "string"
                },
                "unit": {
                  "id": "/vnfd/deployment_flavours/1/assurance_parameters/0/unit",
                  "type": "string"
                }
              },
              "additionalProperties": false
            }
          }
        },
        "additionalProperties": false
      }
    },

    "vnf_lifecycle_events": {
      "id": "/vnfd/vnf_lifecycle_events",
      "type": "array",
      "items": {
        "id": "/vnfd/vnf_lifecycle_events/1",
        "type": "object",
        "properties": {
          "authentication_username": {
            "id": "/vnfd/vnf_lifecycle_events/1/authentication_username",
            "type": "string"
          },
```

```
          "driver": {
            "id": "/vnfd/vnf_lifecycle_events/1/driver",
            "type": "string"
          },
          "authentication_type": {
            "id": "/vnfd/vnf_lifecycle_events/1/authentication_type",
            "type": "string"
          },
          "authentication": {
            "id": "/vnfd/vnf_lifecycle_events/1/authentication",
            "type": "string"
          },
          "authentication_port": {
            "id": "/vnfd/vnf_lifecycle_events/1/authentication_port",
            "type": "integer"
          },
          "vnf_container": {
            "id": "/vnfd/vnf_lifecycle_events/1/vnf_container",
            "type": "string"
          },
          "events": {
            "id": "/vnfd/vnf_lifecycle_events/1/events",
            "type": "object",
            "properties": {
              "start": {
                "id": "/vnfd/vnf_lifecycle_events/1/events/start",
                "type": "object",
                "properties": {
                  "command": {
                    "id": "/vnfd/vnf_lifecycle_events/1/events/start/command",
                    "type": "string"
                  },
                  "template_file": {
                    "id": "/vnfd/vnf_lifecycle_events/1/events/start/template_file",
                    "type": "string"
                  },
                  "template_file_format": {
                    "id":
"/vnfd/vnf_lifecycle_events/1/events/start/template_file_format",
                    "type": "string"
                  }
                },
                "required": [
                  "command",
                  "template_file",
                  "template_file_format"
                ],
                "additionalProperties": false
              },
              "stop": {
                "id": "/vnfd/vnf_lifecycle_events/1/events/stop",
                "type": "object",
                "properties": {
                  "command": {
                    "id": "/vnfd/vnf_lifecycle_events/1/events/stop/command",
                    "type": "string"
                  },
                  "template_file": {
```

```
                    "id": "/vnfd/vnf_lifecycle_events/1/events/stop/template_file",
                    "type": "string"
                  },
                  "template_file_format": {
                    "id":
"/vnfd/vnf_lifecycle_events/1/events/stop/template_file_format",
                    "type": "string"
                  }
                },
                "required": [
                  "command",
                  "template_file",
                  "template_file_format"
                ],
                "additionalProperties": false
              },
              "restart": {
                "id": "/vnfd/vnf_lifecycle_events/1/events/restart",
                "type": "object",
                "properties": {
                  "command": {
                    "id": "/vnfd/vnf_lifecycle_events/1/events/restart/command",
                    "type": "string"
                  },
                  "template_file": {
                    "id": "/vnfd/vnf_lifecycle_events/1/events/restart/template_file",
                    "type": "string"
                  },
                  "template_file_format": {
                    "id":
"/vnfd/vnf_lifecycle_events/1/events/restart/template_file_format",
                    "type": "string"
                  }
                },
                "additionalProperties": false
              },
              "scale-in": {
                "id": "/vnfd/vnf_lifecycle_events/1/events/scale-in",
                "type": "object",
                "properties": {
                  "command": {
                    "id": "/vnfd/vnf_lifecycle_events/1/events/scale-in/command",
                    "type": "string"
                  },
                  "template_file": {
                    "id": "/vnfd/vnf_lifecycle_events/1/events/scale-in/template_file",
                    "type": "string"
                  },
                  "template_file_format": {
                    "id":                    "/vnfd/vnf_lifecycle_events/1/events/scale-
in/template_file_format",
                    "type": "string"
                  }
                },
                "additionalProperties": false
              },
              "scale-out": {
                "id": "/vnfd/vnf_lifecycle_events/1/events/scale-out",
```

```
                "type": "object",
                "properties": {
                  "command": {
                    "id": "/vnfd/vnf_lifecycle_events/1/events/scale-out/command",
                    "type": "string"
                  },
                  "template_file": {
                    "id":                    "/vnfd/vnf_lifecycle_events/1/events/scale-
out/template_file",
                    "type": "string"
                  },
                  "template_file_format": {
                    "id":                    "/vnfd/vnf_lifecycle_events/1/events/scale-
out/template_file_format",
                    "type": "string"
                  }
                },
                "additionalProperties": false
              }
            },
            "required": [
              "start",
              "stop"
            ],
            "additionalProperties": false
          },
          "flavour_id_ref": {
            "id": "/vnfd/vnf_lifecycle_events/1/flavour_id_ref",
            "type": "string"
          }
        },
        "additionalProperties": false
      }
    }

  },
  "additionalProperties": false
}
```

Annex D: PNFD Schema

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "id": "",
  "type": "object",
  "properties": {
       "id": {
      "id": "/vnfd/id",
      "type": "integer"
    },
       "vendor": {
      "id": "/vnfd/vendor",
      "type": "string"
    },
       "descriptor_version": {
      "id": "/vnfd/descriptor_version",
      "type": "string"
    },
        "version": {
      "id": "/vnfd/version",
      "type": "string"
    },
       "name": {
      "id": "/vnfd/name",
      "type": "string"
    },
       "manifest_file_md5": {
          "id": "/vnfd/manifest_file_md5",
          "type": "string"
        },
    "description": {
      "id": "/vnfd/description",
      "type": "string"
    },
    "connection_points": {
          "id": "/nsd/connection_points",
          "type": "array",
          "items": {
            "id": "/nsd/service_deployment_flavours/0",
            "type": "object",
            "properties": {
              "id": {
                "id": "/nsd/service_deployment_flavours/0/id",
                "type": "string"
              },
              "type": {
                "id": "/nsd/service_deployment_flavours/0/flavour_key",
                "type": "string"
              }
                    }
                }
              },
  "additionalProperties": false
  }
```

Annex E: NSD Schema

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "id": "",
  "type": "object",
  "properties": {
    "nsd": {
      "id": "/nsd",
      "type": "object",
      "properties": {
        "id": {
          "id": "/nsd/id",
          "type": "string"
        },
        "name": {
          "id": "/nsd/name",
          "type": "string"
        },
        "vendor": {
          "id": "/nsd/vendor",
          "type": "string"
        },
        "version": {
          "id": "/nsd/version",
          "type": "string"
        },
        "manifest_file_md5": {
          "id": "/nsd/manifest_file_md5",
          "type": "string"
        },
        "vnfds": {
          "id": "/nsd/vnfds",
          "type": "array",
          "items": {
            "type": "string"
          },
          "minItems": 1,
          "uniqueItems": true
        },
        "vnffgd": {
          "id": "/nsd/vnffgd",
          "type": "object",
          "properties": {
            "id": {
              "id": "/nsd/vnffgd/id",
              "type": "string"
            },
            "vendor": {
              "id": "/nsd/vnffgd/vendor",
              "type": "string"
            },
            "version": {
              "id": "/nsd/vnffgd/version",
              "type": "string"
            },
            "ns_version": {
              "id": "/nsd/vnffgd/ns_version",
```

```
              "type": "string"
            },
            "manifest_file_md5": {
              "id": "/nsd/vnffgd/manifest_file_md5",
              "type": "string"
            },
            "vnffgs": {
              "id": "/nsd/vnffgd/vnffgs",
              "type": "array",
              "items": {
                "id": "/nsd/vnffgd/vnffgs/0",
                "type": "object",
                "properties": {
                  "vnffg_id": {
                    "id": "/nsd/vnffgd/vnffgs/0/vnffg_id",
                    "type": "string"
                  },
                  "number_of_endpoints": {
                    "id": "/nsd/vnffgd/vnffgs/0/number_of_endpoints",
                    "type": "integer"
                  },
                  "number_of_virtual_links": {
                    "id": "/nsd/vnffgd/vnffgs/0/number_of_virtual_links",
                    "type": "integer"
                  },
                  "depedent_virtual_links": {
                    "id": "/nsd/vnffgd/vnffgs/0/depedent_virtual_links",
                    "type": "array",
                    "items": [
                      {
                        "id": "/nsd/vnffgd/vnffgs/0/depedent_virtual_links/0",
                        "type": "string"
                      },
                      {
                        "id": "/nsd/vnffgd/vnffgs/0/depedent_virtual_links/1",
                        "type": "string"
                      },
                      {
                        "id": "/nsd/vnffgd/vnffgs/0/depedent_virtual_links/2",
                        "type": "string"
                      }
                    ]
                  },
                  "network_forwarding_path": {
                    "id": "/nsd/vnffgd/vnffgs/0/network_forwarding_path",
                    "type": "array",
                    "items": {
                      "id": "/nsd/vnffgd/vnffgs/0/network_forwarding_path/0",
                      "type": "object",
                      "properties": {
                        "nfp_id": {
                          "id":
"/nsd/vnffgd/vnffgs/0/network_forwarding_path/0/nfp_id",
                          "type": "string"
                        },
                        "graph": {
                          "id": "/nsd/vnffgd/vnffgs/0/network_forwarding_path/0/graph",
```

```
                        "type": "array"
                      },
                      "connection_points": {
                        "id":
"/nsd/vnffgd/vnffgs/0/network_forwarding_path/0/connection_points",
                        "type": "array"
                      },
                      "constituent_vnfs": {
                        "id":
"/nsd/vnffgd/vnffgs/0/network_forwarding_path/0/constituent_vnfs",
                        "type": "array",
                        "items": {
                          "id":
"/nsd/vnffgd/vnffgs/0/network_forwarding_path/0/constituent_vnfs/0",
                          "type": "Object"
                        }
                      }
                    }
                  }
                }
              }
            }
          }
        },
        "lifecycle_events": {
          "id": "/nsd/lifecycle_events",
          "type": "object",
          "properties": {
            "start": {
              "id": "/nsd/lifecycle_events/start",
              "type": "array",
              "items": {
                "id": "/nsd/lifecycle_events/start/0",
                "type": "object",
                "properties": {
                  "vnf_id": {
                    "id": "/nsd/lifecycle_events/start/0/vnf_id",
                    "type": "string"
                  },
                  "vnf_event": {
                    "id": "/nsd/lifecycle_events/start/0/vnf_event",
                    "type": "string"
                  }
                }
              }
            },
            "stop": {
              "id": "/nsd/lifecycle_events/stop",
              "type": "array",
              "items": {
                "id": "/nsd/lifecycle_events/stop/0",
                "type": "object",
                "properties": {
                  "vnf_id": {
                    "id": "/nsd/lifecycle_events/stop/0/vnf_id",
                    "type": "string"
```

```
                },
                "vnf_event": {
                  "id": "/nsd/lifecycle_events/stop/0/vnf_event",
                  "type": "string"
                }
              }
            }
          },
          "scale_out": {
            "id": "/nsd/lifecycle_events/scale_out",
            "type": "array",
            "items": {
              "id": "/nsd/lifecycle_events/scale_out/0",
              "type": "object",
              "properties": {
                "vnf_id": {
                  "id": "/nsd/lifecycle_events/scale_out/0/vnf_id",
                  "type": "string"
                },
                "vnf_event": {
                  "id": "/nsd/lifecycle_events/scale_out/0/vnf_event",
                  "type": "string"
                }
              }
            }
          }
        }
      },
      "vnf_depedency": {
        "id": "/nsd/vnf_depedency",
        "type": "array",
        "items": {
          "type": "string"
        },
        "minItems": 0,
        "uniqueItems": true
      },
      "monitoring_parameters": {
        "id": "/nsd/monitoring_parameters",
        "type": "array",
        "items": {
          "type": "object"
        },
        "minItems": 0
      },
      "vld": {
        "id": "/nsd/vld",
        "type": "object",
        "properties": {
          "id": {
            "id": "/nsd/vld/id",
            "type": "string"
          },
          "vendor": {
            "id": "/nsd/vld/vendor",
            "type": "string"
          },
          "descriptor_version": {
```

```
            "id": "/nsd/vld/descriptor_version",
            "type": "string"
          },
          "manifest_file_md5": {
            "id": "/nsd/vld/manifest_file_md5",
            "type": "string"
          },
          "number_of_endpoints": {
            "id": "/nsd/vld/number_of_endpoints",
            "type": "integer"
          },
          "virtual_links": {
            "id": "/nsd/vld/virtual_links",
            "type": "array",
            "items":
            {
              "id": "/nsd/vld/virtual_links/0",
              "type": "object",
              "properties": {
                "vld_id": {
                  "id": "/nsd/vld/virtual_links/0/vld_id",
                  "type": "string"
                },
                "root_requirements": {
                  "id": "/nsd/vld/virtual_links/0/root_requirements",
                  "type": "string"
                },
                "leaf_requirements": {
                  "id": "/nsd/vld/virtual_links/0/leaf_requirements",
                  "type": "string"
                },
                "qos": {
                  "id": "/nsd/vld/virtual_links/0/qos",
                  "type": "object",
                  "properties": {
                    "none": {
                      "id": "/nsd/vld/virtual_links/0/qos/none",
                      "type": "string"
                    }
                  }
                },
                "test_access": {
                  "id": "/nsd/vld/virtual_links/0/test_access",
                  "type": "string"
                },
                "connections": {
                  "id": "/nsd/vld/virtual_links/0/connections",
                  "type": "array",
                  "minItems": 1
                },
                "connectivity_type": {
                  "id": "/nsd/vld/virtual_links/0/connectivity_type",
                  "type": "string"
                }
              }
            },
            "minItems": 1
          }
```

```
          }
        },
        "service_deployment_flavours": {
          "id": "/nsd/service_deployment_flavours",
          "type": "array",
          "items": {
            "id": "/nsd/service_deployment_flavours/0",
            "type": "object",
            "properties": {
              "id": {
                "id": "/nsd/service_deployment_flavours/0/id",
                "type": "string"
              },
              "flavour_key": {
                "id": "/nsd/service_deployment_flavours/0/flavour_key",
                "type": "string"
              },
              "constituent_vnf": {
                "id": "/nsd/service_deployment_flavours/0/constituent_vnf",
                "type": "array",
                "items": {
                  "id": "/nsd/service_deployment_flavours/0/constituent_vnf/0",
                  "type": "object",
                  "properties": {
                    "number_of_instances": {
                      "id":
"/nsd/service_deployment_flavours/0/constituent_vnf/0/number_of_instances",
                      "type": "integer"
                    },
                    "redundancy_model": {
                      "id":
"/nsd/service_deployment_flavours/0/constituent_vnf/0/redundancy_model",
                      "type": "string"
                    },
                    "vnf_flavour_id_reference": {
                      "id":
"/nsd/service_deployment_flavours/0/constituent_vnf/0/vnf_flavour_id_reference",
                      "type": "string"
                    },
                    "vnf_reference": {
                      "id":
"/nsd/service_deployment_flavours/0/constituent_vnf/0/vnf_reference",
                      "type": "string"
                    }
                  }
                },
                            "minItems": 1
              }
            }
          },
          "minItems": 1
        },
        "auto_scale_policy": {
          "id": "/nsd/auto_scale_policy",
          "type": "object",
          "properties": {
            "criteria": {
              "id": "/nsd/auto_scale_policy/criteria",
```

```
              "type": "array"
            },
            "action": {
              "id": "/nsd/auto_scale_policy/action",
              "type": "string"
            }
          }
        },
        "connection_points": {
          "id": "/nsd/connection_points",
          "type": "array",
          "items": {
            "id": "/nsd/service_deployment_flavours/0",
            "type": "object",
            "properties": {
              "id": {
                "id": "/nsd/service_deployment_flavours/0/id",
                "type": "string"
              },
              "type": {
                "id": "/nsd/service_deployment_flavours/0/flavour_key",
                "type": "string"
              }
                     }
                   }
                },
                "pnfds": {
          "id": "/nsd/pnfds",
          "type": "array",
          "items": {
            "type": "string"
          }
        }
      },
      "required": [
        "id",
        "name",
        "version",
        "vnfds",
        "vnffgd",
        "lifecycle_events",
        "monitoring_parameters",
        "vld",
        "auto_scale_policy"
      ]
    }
  },
  "required": [
    "nsd"
  ]
}
```