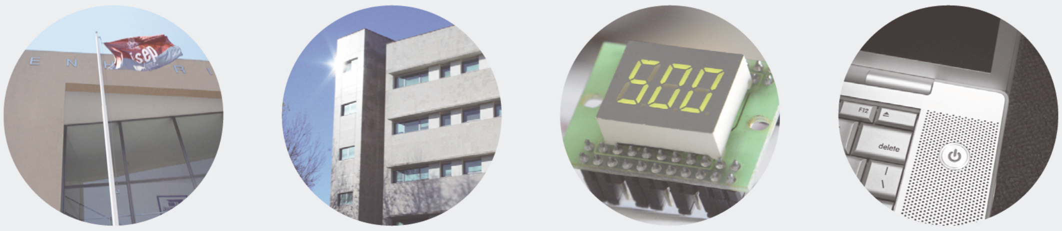
INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA ELECTROTÉCNICA E DE COMPUTADORES



Orquestração de Serviços em Redes 5G

TIAGO FONSECA AMARAL
novembro de 2020

POLITÉCNICO
DO PORTO

Instituto Superior de Engenharia do Porto

Services Orchestration in 5G Networks

Tiago Fonseca Amaral

isep Instituto Superior de Engenharia do Porto

Mestrado em Engenharia Eletrotécnica e de Computadores

Área de Especialização em Telecomunicações

Departamento de Engenharia Eletrotécnica

2020

Relatório elaborado para satisfação parcial dos requisitos da Unidade Curricular de Tese/Dissertação do Mestrado em Engenharia Eletrotécnica e de Computadores.

Candidato: Tiago Fonseca Amaral, n.º 1180440, 1180440@isep.ipp.pt

Orientação científica: Professor Doutor Jorge Mamede, jbm@isep.ipp.pt

Empresa: Altran Portugal, SA

Coorientador: Doutor Sérgio Figueiredo, sergio.figueiredo@altran.com

Coorientador: Mestre Bruno Parreira, bruno.parreira@altran.com

isep Instituto Superior de Engenharia do Porto

Mestrado em Engenharia Eletrotécnica e de Computadores

Área de Especialização em Telecomunicações

Departamento de Engenharia Eletrotécnica

Instituto Superior de Engenharia do Porto

25 de outubro 2020

À minha família.

Agradecimentos

Sem dúvida nenhuma, esta etapa foi a mais desafiante da minha vida académica. O desafio foi proposto e os objetivos foram atingidos com sucesso. Este sucesso deve-se ao trabalho desenvolvido ao longo dos últimos anos, à rotina criada e sobretudo à organização. As pessoas que conheci, ajudaram-me a crescer como pessoa, assim sendo pretendo exprimir a minha gratidão a todos, desde o segurança da escola, aos funcionários do bar, do refeitório, da papelaria, da reprografia e da biblioteca, a todos os professores, aos meus colegas de curso, ao Departamento de Engenharia Eletrotécnica (DEE), à associação de estudantes do Instituto Superior de Engenharia do Porto (ISEP), à ISEP Academy, ao ISEP e à empresa Altran Portugal por me permitir integrar num projeto tão desafiante com uma equipa de trabalho altamente qualificada.

Um enorme obrigado à minha família e aos meus amigos pela motivação e apoio que me deram para continuar a lutar pelos meus objetivos académicos e pessoais.

Agradeço aos meus colegas de curso, de uma forma especial, pelos momentos de estudo e trabalho em conjunto, bem como a amizade criada e respeito mútuo. E ainda a todos os outros colegas que conheci na minha passagem pelo ISEP.

Ao Professor Doutor Jorge Mamede, agradeço pelo seu apoio, disponibilidade, orientação científica no desenvolvimento desta dissertação, e também pelos ensinamentos prestados ao longo do curso.

Ao Doutor Sérgio Figueiredo e ao Mestre Bruno Parreira, coorientadores e membros do projeto Mobilizador 5G na Altran Portugal, agradeço pela integração, apoio e orientação prestada no projeto desenvolvido. Também agradecer à minha equipa de trabalho, nomeadamente, aos meus colegas de equipa, ao Ricardo, ao Paulo, ao João, ao Bruno e ao Diogo, pelo espírito de equipa criado e pela passagem de inúmeros conhecimentos que fui adquirindo nesta área.

Por fim, gostaria de agradecer a todas as pessoas que direta ou indiretamente contribuíram para que a realização deste projeto fosse possível, manifesto gratidão a todas essas pessoas.

Abstract

The fifth-generation of mobile communications (5G) introduces significant changes in the deployment of networking infrastructure, based on fundamental pillars like Network Slicing, Multi-Access Edge Computing (MEC), Network Functions Virtualization (NFV) and Software-Defined Networking (SDN). This radical transformation in its architecture brings several challenges and puts under pressure the telecommunications operators to keep competitive Quality of Service (QoS) levels and enhance the way network services are designed, deployed and managed.

The ability to orchestrate and manage the network assumes a crucial role in maximizing the advantages related to the use of these technologies and architectures. In this sense, operators need to use open-source developments, avoiding technology and vendor lock-in, reducing operating costs and time-to-market for new products. Several open-source network orchestration solutions have appeared with the aim of proposing a complete orchestration framework. In this context, the Open Network Automation Platform (ONAP) emerged as a very valuable open-source platform, both as a standards-aligned and standards-influencing solution capable of enabling end-to-end management and orchestration of services and resources through a multi-domain infrastructure.

This Dissertation aims to integrate services developed for 5G networks, by enabling essential lifecycle management operations (e.g., instantiation, termination, etc.) using a reference open-source orchestration platform.

In order to validate the considered orchestration platform, a virtual Content Delivery Network (vCDN) service was integrated. To depict the platform closed-loop control capabilities, a scenario of congestion was proposed and handled through the scale-out operation of the vCDN service. The quality of the designed workflow was improved in two specific operations of the vCDN service lifecycle management.

Keywords: 5G, Network Slicing, NFV, ONAP, Orchestration.

Resumo

A quinta geração de comunicações móveis (5G) introduz mudanças significativas na implementação da infraestrutura de rede, sendo baseada em pilares fundamentais como *Network Slicing*, *Multi-Access Edge Computing* (MEC), *Network Functions Virtualization* (NFV) e *Software-Defined Networking* (SDN). Esta transformação radical na sua arquitetura traz diversos desafios e coloca as operadoras de telecomunicações sob pressão para manter níveis competitivos de qualidade de serviço e aprimorar o modo como os seus serviços de rede são projetados, implementados e geridos.

A capacidade de orquestrar e gerir a rede assume um papel crucial para maximizar as vantagens relacionadas com a utilização destas tecnologias e arquiteturas. Neste sentido, as operadoras precisam de utilizar projetos *open-source*, evitando assim *hardware* e *software* proprietário, reduzindo os custos operacionais e também o tempo de colocação no mercado de novos produtos. Várias soluções de orquestração de rede *open-source* surgiram com o objetivo de propor uma plataforma de orquestração completa. Neste contexto, o *Open Network Automation Platform* (ONAP) emergiu como uma plataforma *open-source* muito valiosa, tanto como uma solução orientada a padrões e que influencia os padrões, capaz de permitir a gestão e orquestração *end-to-end* dos serviços e recursos através de uma infraestrutura multi-domínio.

Esta Dissertação tem como objetivo a integração de serviços desenvolvidos para as redes 5G, de forma a permitir as operações essenciais da gestão do seu ciclo de vida (p. ex., instanciação, terminação, etc.), utilizando uma plataforma de orquestração *open-source* de referência.

Para a validação da plataforma orquestração proposta foi realizada a integração de um serviço *virtual Content Delivery Network* (vCDN). A descrição das funcionalidades da metodologia de *closed-loop control* da plataforma de orquestração, teve por base a definição de um cenário de congestionamento, que foi resolvido através da operação de *scale-out* do serviço vCDN. A qualidade do *workflow* desenvolvido foi aprimorada em duas operações específicas de gestão do ciclo de vida do serviço vCDN.

Palavras-Chave: 5G, *Network Slicing*, NFV, ONAP, Orquestração.

Contents

List of Figures

List of Tables

List of Acronyms

3GPP	Third Generation Partnership Project
4G	Fourth-Generation of mobile telecommunications technology
5G	Fifth-Generation of mobile telecommunications technology
5G-PPP	5G - Public Private Partnership
5GS	5G System
AAI	Active and Available Inventory
AMF	Access and Mobility Management Function
API	Application Programming Interface
APP-C	Application Controller
AR/VR	Augmented & Virtual Reality services
ARIB	Association of Radio Industries and Businesses
ATIS	Alliance for Telecommunications Industry Solutions
AWS	Amazon Web Services
BPD	Business Process Diagram
BPEL	Business Process Execution Language
BPM	Business Process Management
BPMN	Business Process Model and Notation
BSS	Business Support Systems
CC	Critical Communications
CCSA	China Communications Standards Association
CDN	Content Delivery Network
CLAMP	Closed Loop Automation Management Platform
CMMN	Case Management Model and Notation
CP	Control Plane
CPU	Central Processing Unit
CSAR	Cloud Service ARchive
CSP	Communication Service Providers

D-CPI	Data-Controller Plane Interface
DCAE	Data Collection, Analytics and Events
DMaaP	Data Movement-as-a-Platform
DP	Data Plane
EM	Element Manager
ETSI	European Telecommunications Standards Institute
FCAPS	Fault-management, Configuration, Accounting, Performance, Security
GB	Gigabyte
GUI	Graphical User Interface
GVNFM	Generic Virtualized Network Function Manager
HD	Hard Disk
HOT	Heat Orchestration Template
HTTP	Hypertext Transfer Protocol
ID	Identifier
IDE	Integrated Development Environment
IMT	International Mobile Telecommunications
IP	Internet Protocol
IPR	Intellectual Property Rights
ISEP	*Instituto Superior de Engenharia do Porto*
ISG	Industry Specification Group
ISO	International Organization for Standardization
IT	Information Technology
ITU	International Telecommunication Union
ITU-R	ITU - Radiocommunications Sector
ITU-T	ITU - Telecommunication Standardization Sector
IaaS	Infrastructure-as-a-Service
IoT	Internet of Things
JSON	JavaScript Object Notation
JVM	Java Virtual Machine
KPI	Key Performance Indicator
LTE	Long-Term Evolution
MANO	Management and Orchestration
MEC	Multi-access Edge Computing
MEF	Metro Ethernet Forum
MSB	Microservices Bus

NBI	NorthBound Interface
NF	Network Function
NFV	Network Function Virtualization
NFVI	Network Function Virtualization Infrastructure
NFVO	Network Function Virtualization Orchestrator
NG-RAN	Next Generation - Radio Access Network
NGMN	Next Generation Mobile Networks Alliance
NS	Network Service
NSD	Network Service Descriptor
NSO	Network Service Orchestration
NaaS	Network-as-a-Service
OMG	Object Management Group
ONAP	Open Network Automation Platform
ONF	Open Networking Foundation
OODA	Observe, Orient, Decide, Act
OOM	ONAP Operations Manager
OS	Operating System
OSI	Open System Interconnection
OSM	Open Source Management and Orchestration
OSS	Operations Support System
PBNM	Policy-Based Network Management
PLMN	Public Land Mobile Network
PNF	Physical Network Functions
QoS	Quality of Service
RAM	Random Access Memory
REST	Representational State Transfer
RESTful	Representational State Transfer conforming Web services
SDC	Service Design and Creation
SDN	Software-Defined Network
SDN-C	Software-Defined Network - Controller
SDO	Standards Developing Organization
SLA	Service Level Agreement
SMF	Session Management Function
SO	Service Orchestrator
TCP	Transmission Control Protocol

TLS	Transport Layer Security
TMF	TeleManagement Forum
TOSCA	Topology and Orchestration Specification for Cloud Applications
TSDSI	Telecommunications Standards Development Society, India
TTA	Telecommunications Technology Association
TTC	Telecommunications Technology Committee
UML	Unified Modeling Language
UP	User Plane
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
UUID	Universally Uniquef Identifier
VF	Virtual Function
VF-C	Virtual Function Controller
VFC	Virtual Function Component
VID	Virtual Instantiation Deployment
VIM	Virtualized Infrastructure Manager
VM	Virtual Machine
VNF	Virtualized Network Function
VNFD	Virtualized Network Function Descriptor
VNFM	Virtual Network Function Manager
VSP	Vendor Software Products
WS-BPEL	Web Services - Business Process Execution Language
XML	Extensible Markup Language
YAML	Yet Another Markup Language
YANG	Yet Another Next-Generation
eMBB	enhanced Mobile Broadband
eV2X	enhanced Vehicle-to-Everything
mMIoT	massive Internet of Things
mMTC	massive Machine-Type Communications
uRLLC	ultra-Reliable and Low Latency Communications
vCDN	virtual Content Delivery Network
vCPU	virtual Central Processing Unit

Chapter 1

Introduction

1.1 Context

Mobile wireless communications have become an essential part of our society with increasing importance over the last years. Some services deeply rooted in our daily life, such as education, banking, shopping, and public services, are already available online, providing new levels of comfort and efficiency accessible to the whole society [1].

The fifth-generation of mobile communications (5G) has been triggered by the increasingly strong demand for a well-connected society, which is leveraging new services and business models in different markets. This generation is the first mobile communications' system that is by design addressing a wide range of needs of vertical industries, such as manufacturing, energy, healthcare, automotive, and multimedia, through an unlimited mobile broadband experience, provides massive connectivity for everything from human-held smart devices to sensors and machines, and most importantly, it has the ability to support critical machine communications with instant action and ultra-high reliability [2].

The 5G introduces significant changes in the deployment of networking infrastructure, based on fundamental pillars like Network Slicing, Multi-Access Edge Computing (MEC), Network Functions Virtualization (NFV) and Software-Defined Networking (SDN). Network operations and services are becoming Cloud-enabled in almost all sectors of industry, including also the telecommunications industry. For this reason, it will be possible the creation of new opportunities to generate value in the era of autonomous driving, Cloud computing, Internet of Things (IoT), augmented and virtual reality (AR/VR) services. The main domains of the 5G system are applications, Cloud, transport, wireless access and management, including orchestration [3].

The ability to orchestrate and manage the network assumes a crucial role in maximizing the advantages related to the use of these technologies and architectures. In this sense, operators need to use open-source developments, avoiding technology and vendor lock-in, reducing operating costs and time-to-market for new products.

1.2 Motivation

In parallel to the improvements made in 5G networks, a number of open-source Network Service Orchestration (NSO) solutions, based on a software-driven approach, have appeared with the aim of allowing telecommunications operator to manage, module and automate their networks and services, with a complete orchestration framework.

In this context, the Open Network Automation Platform (ONAP) emerged as a very valuable open-source project, both as a standards-aligned and standards-influencing solution for the support of multi-domain and end-to-end service orchestration and automation.

In Portugal, the 5G Mobilizer project aims to be an instrument for the development and innovation of 5G technology covering all functional domains: access, core, and vertical sectors. Within the scope of the project, all products from all domains will be integrated, validated, tested, and demonstrated in an ecosystem that promotes new 5G services, which will benefit from a realistic test environment. This project has the participation of several industrial entities, Altran included, with activity in the telecommunications sector and also counts with the collaboration of four entities of the national scientific and technological system. Apart from other contributions to this project, Altran is developing and testing an open-source NSO solution to orchestrate and automate services and resources in 5G networks [4].

1.3 Objectives

This Dissertation aims to integrate services developed for 5G networks, by enabling essential lifecycle management operations (e.g., instantiation, termination, etc.) using a reference open-source orchestration platform. In order to address the proposed objective, the following stages were defined:

- Study and analysis on important concepts about 5G networks, Content Delivery Network (CDN) services, orchestration, open-source NSO solutions and there modeling languages;

- Study and practice on the Camunda Business Process Management (BPM) platform, including design and execution of workflows;

- Study and comprehension of the orchestration platform and their operations;

- Integration of a 5G service with the orchestration platform;

- Design, implementation and validation of service lifecycle management operations;

- Tests and analysis for improving the quality of the designed workflow.

1.4 Contributions

The contributions of this work include:

- The design and development of the scale-out operation of a 5G service, which comprises lifecycle management functions, such as the creation, instantiation, etc.
- The validation of the considered orchestration platform in a 5G network congestion scenario.

The development of this Dissertation contributed to the 5G Mobilizer project.

1.5 Document Structure

The structure of this Dissertation is organized as follows:

- Chapter 1 – introduces core topics with the context, motivation, goals and contributions of this work;
- Chapter 2 – provides the essential concepts and key enablers of the 5G networks as well as an overview of the orchestration area along with existing solutions.
- Chapter 3 – presents the used orchestration solution and the definition of a Use Case scenario;
- Chapter 4 – describes the implementation environment and design process stages, and also detailed information about the operation design elements;
- Chapter 5 – describes the operation design to orchestrate the proposed Use Case and highlights key activities that are performed;
- Chapter 6 – provides the tests performed and the analysis of the obtained results;
- Chapter 7 – presents the conclusions and the future work.

Chapter 2

State of the Art

This Chapter presents a review of the essential background concepts and the state of the art. It begins, in Section 2.1, with a brief overview about the evolution of mobile networks and the presentation of 5G technology, including its standardization, usage scenarios, requirements, architecture, and key enablers. The Section 2.2 comprises the main concepts of orchestration and some advances in orchestration mechanisms, with special attention to the Closed-loop Control theme presented in subsection 2.2.3.1. Following, Section 2.3 summarizes and presents open-source NSO solutions for network automation. Then, in Section 2.4 the principal modeling languages and its value for process orchestration are presented. The Section 2.5 introduces BPM tools for workflow and process automation. Finally, Section 2.6 presents the role of CDN in the distribution of digital content.

2.1 Fifth-Generation of Mobile Communications

The history of mobile communications technologies is divided into generations with the first generation being characterized by the analog mobile radio systems, the second generation was the first with digital mobile systems, and the third generation was the first mobile system handling broadband data. The fourth-generation (4G) or Long-Term Evolution (LTE), provides even better support for mobile broadband as described in [5].

The latest generation of mobile communications being implemented is 5G, which is projected to meet diverse and stringent requirements that are currently not supported by current mobile networks, like ubiquitous connectivity (connectivity available anywhere), high-speed connection (ten times higher than 4G) and zero latency (lower than few milliseconds) [6]. This "revolution" is catalyzed by the way that society creates, shares and consumes information. The main drivers behind the anticipated traffic growth are Device proliferation (volume of devices connected), Video usage (e.g., video-on-demand services) and Application uptake [7].

2.1.1 Standardization

The process of defining a standard for mobile communication is an iterative and ongoing process. The standardization process of 5G mobile networks is defined by various entities that work together contributing to the creation of technical specifications and standards as well as regulation in the mobile communications area. These can be divided into three groups: Standards Developing Organizations (SDO), regulatory bodies and administrations, and industry forums [8].

The International Telecommunication Union (ITU) is a standardization group that coordinates the contributions of industry, government, and private sector in the development of global broadband multimedia International Mobile Telecommunication system (IMT). IMT is a nomenclature used by the ITU community to designate broadband mobile systems. It encompasses IMT-2000 (3G), IMT-Advanced (4G) and IMT-2020 (5G) collectively [9]. ITU defined the process of evaluation and the subsequent selection of mobile technologies that fulfill several established technical parameters (peak data rate, latency, spectrum efficiency, etc.). In this regard, IMT-2020 are mobile systems that include the new capabilities of IMT that go beyond those of IMT-Advanced which would make IMT-2020 more efficient, fast, flexible, and reliable when providing diverse services in the intended usage scenarios [10], presented in Section 2.1.2.

The ITU-Radiocommunications Sector (ITU-R) develops and adopts the international regulations on the use of the radio frequency spectrum. IMT-2020 represents the latest deployment on the standardization of 5G globally, contributing mainly to the standardization timeline and the requirements of 5G mobile networks, both widely adopted by most of the standardization bodies and forums [11].

The Third Generation Partnership Project (3GPP) is the main organism that develops technical specifications for advanced mobile communications including 5G. These technical specifications are then transposed into standards by the seven regional SDO that form the 3GPP partnership (as shown in Figure 2.1), in Europe (ETSI), India (TSDSI), Korea (TTA), China (CCSA), Japan (ARIB and TTC), the United States (ATIS). The 3GPP also receives a contribution from several entities, especially highlighted for Next Generation Mobile Networks Alliance (NGMN) that provided the initial concept of network slicing, and 5G Public-Private Project (5G-PPP), in Europe, that have an important role in architectural aspects of 5G [2, 8, 12, 13]. The regional SDO is also responsible for establishing and enforcing an Intellectual Property Rights (IPR) policy [14].

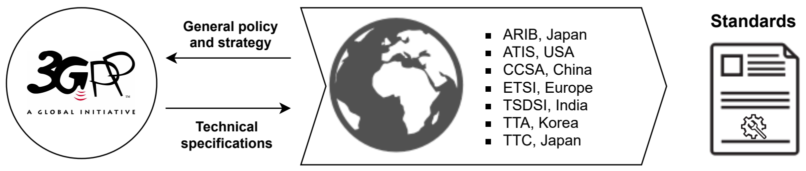


Figure 2.1: Regional SDO (adapted from [14]).

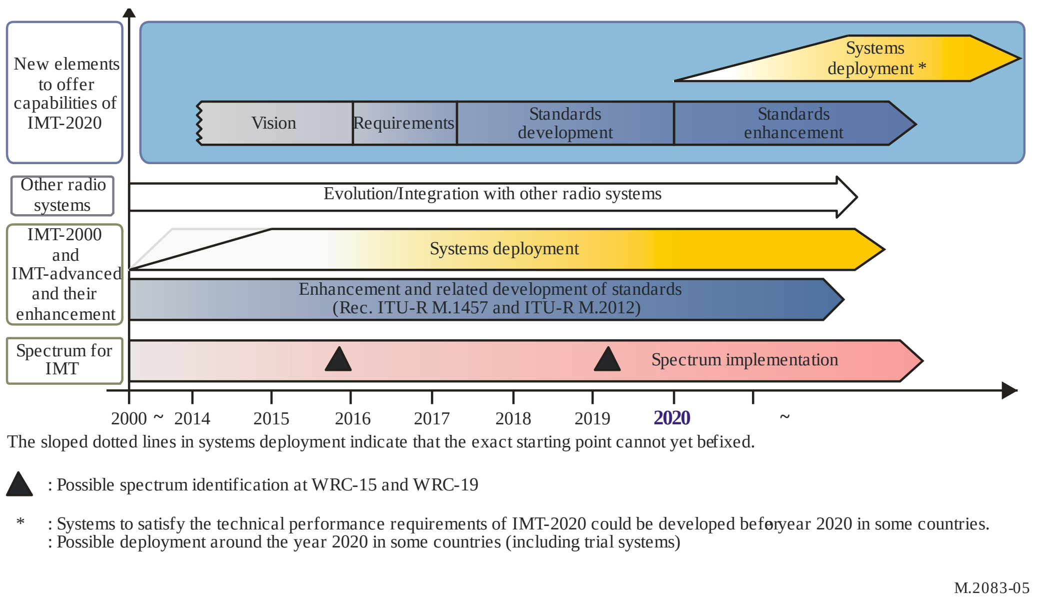

New elements to offer capabilities of IMT-2020
Systems deployment *
Vision
Requirements
Standards development
Standards enhancement
Other radio systems
Evolution/Integration with other radio systems
IMT-2000 and IMT-advanced and their enhancement
Systems deployment
Enhancement and related development of standards (Rec. ITU-R M.1457 and ITU-R M.2012)
Spectrum for IMT
Spectrum implementation
2000 ~ 2014
2015
2016
2017
2018
2019
2020
~
The sloped dotted lines in systems deployment indicate that the exact starting point cannot yet befixed.
: Possible spectrum identification at WRC-15 and WRC-19
* : Systems to satisfy the technical performance requirements of IMT-2020 could be developed beforeyear 2020 in some countries.
: Possible deployment around the year 2020 in some countries (including trial systems)
M.2083-05

Figure 2.2: Phases and expected timelines for IMT-2020 by ITU-R [11].

SDO develop and agree on technical standards for mobile communications systems, in order to provide interoperability between those products and make it possible for the industry to produce and deploy standardized products [8]. 3GPP introduces new services and new features via releases. Every release is staggered and work is done on multiple Releases in parallel at different stages [14]. At the moment, 3GPP is finalizing work on Release-16 and work is already well in progress on Release-17 (Stage 1) as stated in [15].

The timelines associated with IMT-2020 depend on several factors, e.g., technical capabilities and technology development, standards development and their enhancement, spectrum deployment, etc. [11]. These factors are interrelated as depicted in Figure 2.2. The 3GPP standards for 5G are expected to become available for commercial deployment and service delivery between 2020 and 2030 [15].

2.1.2 Use Cases and Requirements

As stated by 3GPP in the specification TS 22.891 [16] and by ITU-R in the recommendation M.2083-0 [11], the requirements of 5G usage scenarios have been defined and are categorized into the following groups:

- Massive Internet of Things (mIoT) by 3GPP and massive Machine Type Communications (mMTC) by ITU – are characterized by a huge number and wide variety of connected devices (e.g., sensors and wearables), commonly transmitting a relatively low volume of non-delay sensitive data. Devices are required to be low cost, and have a very long battery life. It comprises use cases, such as smart home and city, smart utilities, e-Health, and smart wearables. More information about this category can be found in [17].

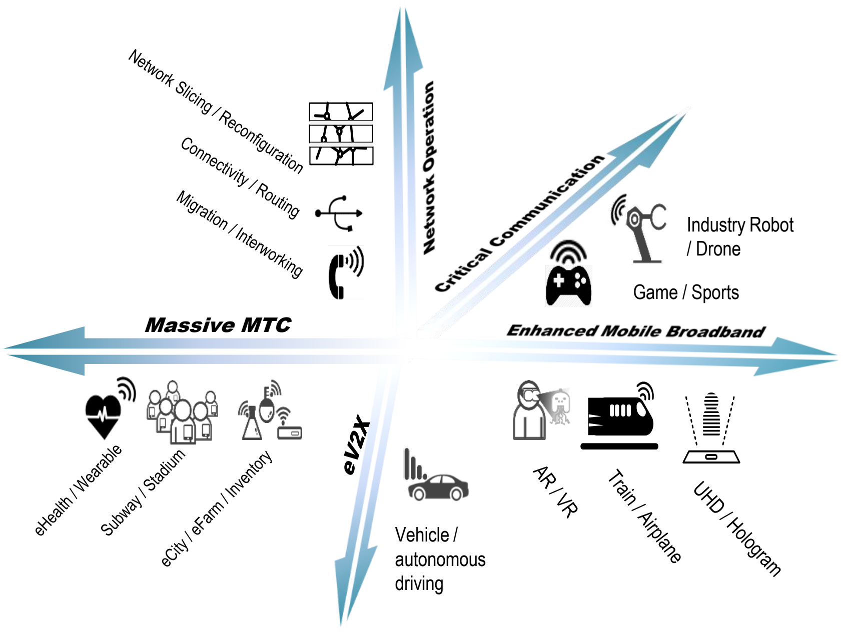


Figure 2.3: Categories of different use cases defined by 3GPP [16].

- Critical Communications (CC) by 3GPP and ultra-Reliable and Low Latency Communications (uRLLC) by ITU – has highly rigorous requirements for capabilities, such as availability, latency, reliability, and throughput. Some examples in [18], include wireless control of industrial manufacturing or production processes, remote medical surgery, distribution automation in a smart grid, transportation safety, etc.

- Enhanced Mobile Broadband (eMBB) by 3GPP and ITU – addresses the human-centric use cases for access to services, multimedia content, and data. This category is decomposed into five sub-categories defining High Data Rates, Higher Density, Deployment and Coverage, Higher User Mobility and Devices with highly variable data rates. The requirements analysis for this usage can be found in [19].

- Network Operation by 3GPP – addresses the functional system requirements including aspects, such as flexible functions and capabilities, new value creation, scalability, network slicing, efficient content delivery, optimizations and enhancements, migration, interworking, and security. Further details are available in [20].

- Enhancement of Vehicle-to-Everything (eV2X) by 3GPP – this use case group demands low latency and high-reliability communications to support real-time response to avoid road accidents. It comprises autonomous driving, safety and non-safety aspects associated with the vehicle [21].

The conceptual diagram, illustrated in Figure 2.3, represents the directions of service improvements proposed in the technical reports from 3GPP.

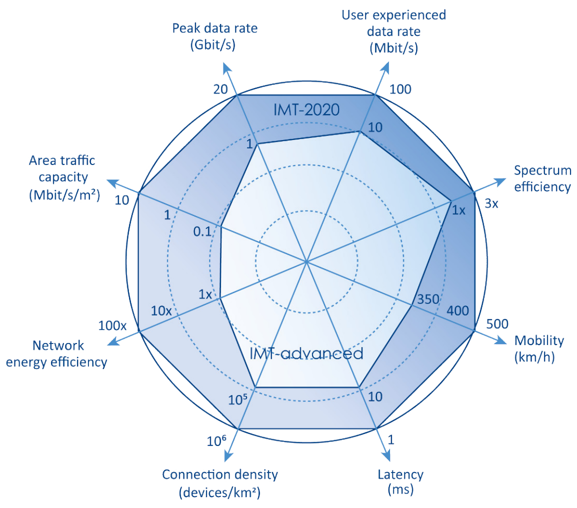


Figure 2.4: Enhancement of key capabilities from IMT-Advanced to IMT-2020 [23].

In the 3GPP specification TS 22.891 [16] several use cases have been identified. The implementation of 5G network will originate new use cases and also new services and business models, mainly empowering the support of various sectors of the industry, referred to as "verticals", e.g., Automotive, Smart Cities, e-Health, Public Safety, etc. [22].

To meet the requirements previously mentioned, a broad variety of capabilities, tightly coupled with intended usage scenarios and applications for IMT-2020 is envisioned. ITU-R defines, in [11], eight Key Performance Indicators (KPI). The key design principles are flexibility and diversity to serve many different use cases and scenarios, for which the capabilities of IMT-2020, includes KPI for metrics, such as e.g., latency, mobility, etc.

As stated, 5G should deliver significantly increased operational performance (e.g., increased spectral efficiency, higher data rates, low latency), as well as superior user experience (near to fixed network but offering full mobility and coverage). Also needs to supply the deployment of mIoT, while still offering acceptable levels of equipment cost, energy consumption, network deployment, and operation cost. It needs to support a wide variety of applications and services [23]. The comparison of key capabilities between IMT-Advanced (4G) and IMT-2020 (5G), is illustrated in Figure 2.4 according to ITU-R M.2083-0 [23].

2.1.3 Architecture

In order to meet the requirements presented in Section 2.1.2, it is essential that the network should be architected for inherent scalability, versatility, and performance to cost-effectively.

The architecture for 5G System (5GS) was accomplished by 3GPP in the specification TS 23.501 [24] which covers the study "System Architecture for the 5G System" and in the specification TS 38.801 [25] which is focused on the study of "RAN Architecture and Interfaces" from an Access Architecture and Interface specification point of view.

The specification TS 23.501 includes the definition of scenarios in all aspects (both roaming and non-roaming), mobility within 5GS, Quality of Service (QoS), policy control and charging, etc. [24]. In the specification TS 38.801, the study item "New Radio Access Technology" is addressed [25].

A service-based architecture was adopted for the 5GS, which brings improved radio units known as Next-Generation Radio Access Network (NG-RAN) for 5G Access Network, the introduction of network slicing concept and the separation of Control Plane (CP) and Data Plane (DP) for 5G Core Network, that will require a Cloud-native design to deliver 5G services and a strong reliance on virtualized functions [26].

The 5GS architecture is projected to support services and data connectivity, this implies major changes in the implementation and deployments to use techniques based on Network Functions Virtualization (NFV) and Software-Defined Networking (SDN) [24], such concepts are covered later in Section 2.1.4.1 and 2.1.4.2, respectively. In the following subsections, a brief description of the model and concepts of architecture is carried out.

2.1.3.1 Service Based Architecture

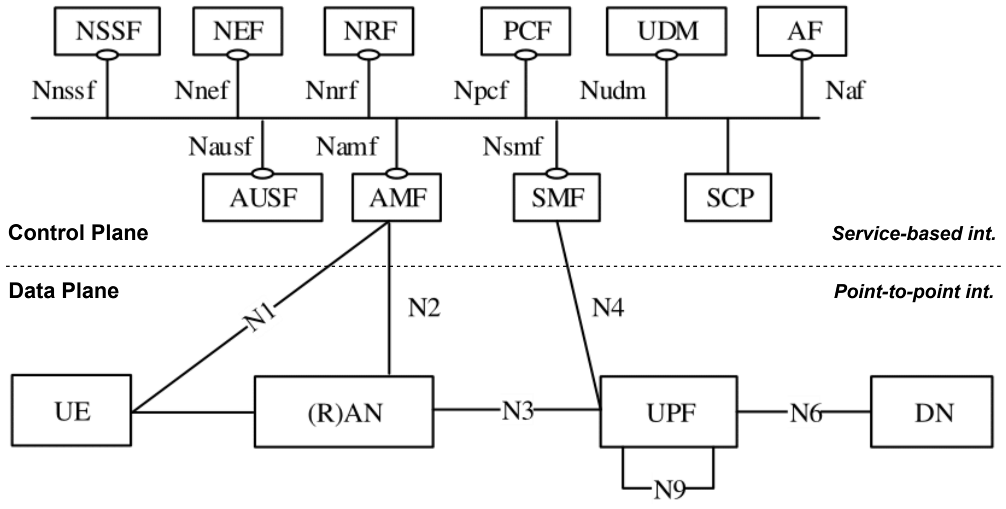
The 5GS architecture is defined as a service-based architecture, i.e., a system architecture in which the system functionality is achieved by a set of Network Functions (NF) providing services to other authorized NF to access their services. The interaction between NF is represented in two ways by 3GPP in the specification TS 23.501 [24]:

- A service-based representation, where NF (e.g., Access and Mobility Management Function (AMF)) within the CP enables other authorized NF to access their services. This representation also includes point-to-point reference points where necessary;

- A reference point representation, shows the interaction that exist between the NF services in the NF described by point-to-point reference point between any two NF (e.g., AMF and Session Management Function (SMF)).

The architecture of 5GS is represented in Figure 2.5, in which the service modularity breaks down into the following self-contained functional components communicating via standardized lightweight interfaces (e.g., Namf, Nnef, etc.) [24, 27].

A service-based interface expresses how the set of services is exposed or provided by a given NF, which employs a well-defined Representational State Transfer (REST) interface using HTTP/2 over TCP/TLS with JavaScript Object Notation (JSON) bodies. The NF service operations are invoked through these interfaces. NF are independent, self-contained, and reusable. The independent scaling of the capacity of CP and DP is achieved by their separation. For example, if more CP capacity is needed, it should be straightforward to add it without affecting the DP of the network. Further information about this category can be found in [28].



Legend:

AMF	- Access and Mobility Management Function	PCF	- Policy Control Function
AF	- Application Function	(R)AN	- Radio Access Network
AUSF	- Authentication Server Function	SCP	- Service Communication Proxy
DN	- Data Network (e.g., Internet access)	SMF	- Session Management Function
NEF	- Network Exposure Function	UDM	- Unified Data Management
NRF	- Network Repository Function	UE	- User Equipment
NSSF	- Network Slice Selection Function	UPF	- User Plane Function

Figure 2.5: 5GS architecture by 3GPP in TS 23.501 (adapted from [24]).

2.1.3.2 Network Slicing

The network slicing concept was introduced by NGMN in [29]. In the context of 5G networks, this concept allows the telecommunications operators to provide different services, with each service having its own performance requirements, over the same physical infrastructure.

In accordance with the specification TS 23.501 [24] of 3GPP, the network slicing is being defined as a logical end-to-end network that can be dynamically created. Network slicing facilitates multiple logical self-contained networks on top of a common physical infrastructure platform enabling a flexible stakeholder ecosystem that allows business and technical innovation integrating physical and/or logical network and Cloud resources into a programmable, open software-oriented multi-tenant network environment [30]. A network slice instance is defined within a Public Land Mobile Network (PLMN) and must include the Core Network, the CP and DP network functions as well as the 5G Access Network.

The concept of 5G network slicing is illustrated in Figure 2.6, with different scenarios running on a common underlying multi-vendor and multi-domain virtualized networking environments. Each slice consists of a set of virtual NF that run on the same physical infrastructure independently managed and addresses a particular use case [31, 32].

Network slicing is a complex concept that is built on top of other main concepts, like e.g., Automation, Isolation, Customization, Elasticity, Programmability, etc., such concepts are described in [30].

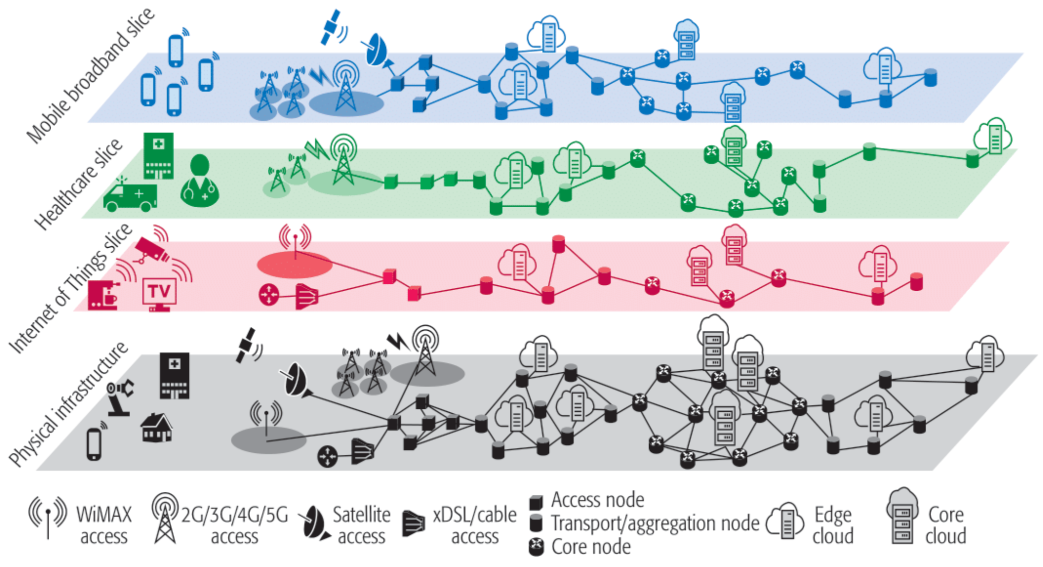


Figure 2.6: 5G Network slicing concept with different scenarios [31].

The 5G network slicing enablers are NFV, SDN, Multi-access Edge Computing (MEC), Cloud/Fog Computing, Network Hypervisors, Docker Containers and Virtual Machines (VM) [6].

2.1.3.3 Next-Generation Radio Access Network

To meet the requirements presented in the use cases and to exploit the potential of new technologies, 3GPP is redesigning and optimizing the new radio-access technology for inherent efficiency, flexibility, and scalability [8].

The NG-RAN is an innovative air interface called New Radio (NR), which is designed primarily for new spectrum bands [25]. An overall description of the NG-RAN is addressed in [33] and an overview of historical and legacy RAN architectures is provided in [34]. Further details about NR Physical Layer can be found on 3GPP specifications TS 38.201 [35], TS 38.211 [36] and TS 38.212 [37]. Although relevant, these 5G characteristics are out of the purpose of this Dissertation.

2.1.4 Key Enablers

In a digital world, the innovation cycles intensify and demand greater flexibility and dynamism than hardware-based devices allow. A hard-wired network with unique functions boxes is hard to maintain, slow to evolve, and prevent Communication Service Providers (CSP) from offering dynamic services [38, 39].

As mentioned in Section 2.1.3, 5G contemplates an architecture that leverages the structural separation of hardware and software, supported by network programmability. Network slicing concept is only possible when resources, virtualization, orchestration and isolation of the network are accomplished [31]. Network operators will be able to deploy, create and manage their services, due to the technological advances in NFV and SDN.

The main enablers like NFV, SDN, Telco Cloud and MEC are encompassed in the following subsections.

2.1.4.1 Network Function Virtualization

NFV is standardized by the European Telecommunications Standards Institute (ETSI), which is a 3GPP partner in Europe. The network operators have started an initiative on NFV and Industry Specification Group (ISG) was founded by ETSI, a group in charge of developing requirements and architecture for virtualization of various functions within telecoms networks [38, 39]. The ETSI ISG NFV helps by setting requirements and architecture specifications for hardware and software infrastructure needed to make sure virtualized functions are maintained, and also manages guidelines for developing NF [39].

NFV is a network architecture to transform the paradigm of the way that network operators manage, deploy, and architect networks. NFV aims at reinforcing the diversity of network devices onto industry-standard high-volume servers. These servers can be located at the different network nodes as well as end-user premises.

In this context, NFV relies upon but differs from traditional server virtualization. Unlike server virtualization, Virtualized Network Functions (VNF) may consist of one or more VMs running different software and processes in order to replace custom hardware appliances. Multiple VNF are to be used in sequence with the aim to provide meaningful services to the customer [38]. NFV decouples NF, such as routing, intrusion detection, firewalls from proprietary hardware platforms and implements these functions in software. It utilizes standard virtualization technologies that lead on high-performance hardware to virtualize NF. It is applicable to any DP processing or CP function in both wired and wireless network infrastructures [40, 41]. ETSI specification GS NFV 002 [42] describes the major differences in the virtualized network service provisioning is realized in comparison to current practice. The NFV follows three base principles:

- **Decoupling software from hardware:** as the software elements of a NF and their hardware counterparts are no longer bound, their evolution can happen independently, which enables the software to progress separately from the hardware, and vice versa;

- **Dynamic operation:** the decoupling of the functionality of the NF into instantiable software components provides greater flexibility to scale the actual VNF performance more dynamically and with finer granularity;

- **Flexible network function deployment:** the detachment of software components from hardware enables more efficient use of the virtualized infrastructures, allowing for a more automated and faster instantiation of NF and VNF over the same physical platform.

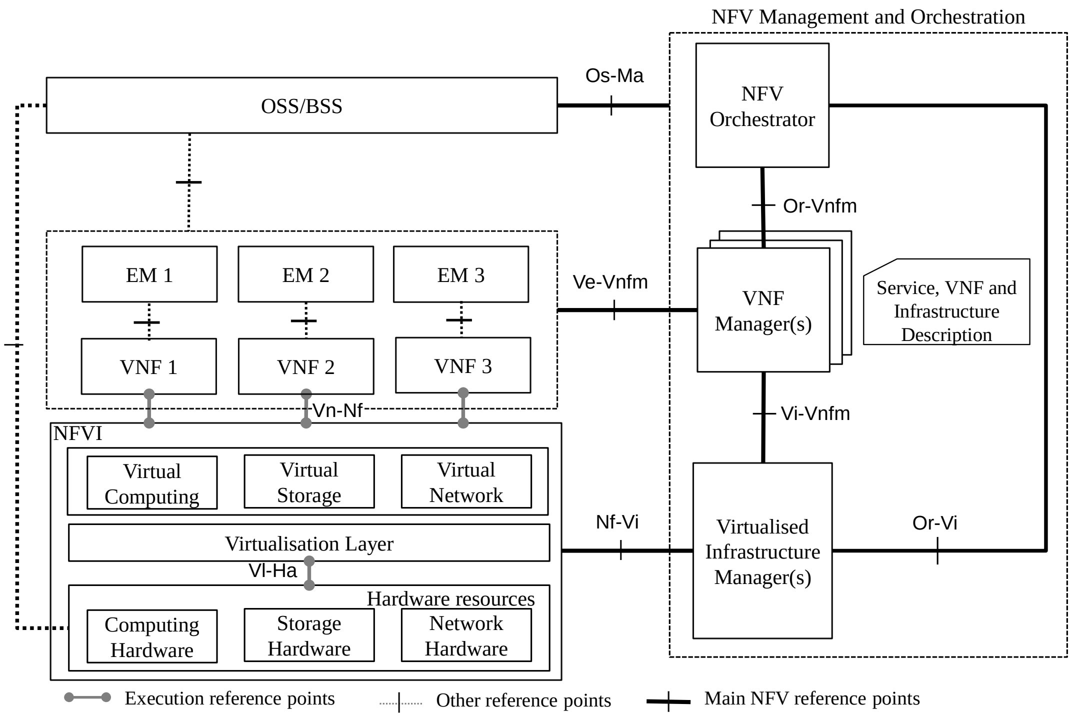


Figure 2.7: ETSI NFV reference architecture [42].

This architecture, from a high-level perspective, is organized into three main functional domains [42]:

- **VNF:** comprises the software implementation of a NF, such as routing, etc., which is capable of running over the NFV Infrastructure (NFVI);
- **NFVI:** includes the different physical resources and how can be virtualized, and supports the execution of the VNFs;
- **NFV Management and Orchestration (MANO):** covers lifecycle management and the orchestration of physical and/or software resources that support the infrastructure virtualization, and the lifecycle management of VNFs. It also focuses on all virtualization-specific management tasks necessary in the NFV framework.

The NFV reference architectural framework, depicted in Figure 2.7, is formed by three main areas (VNF, NFVI, and NFV-MANO), along with the functional blocks and reference points. In the following list, important functional blocks for the management and orchestration operations are described [42]:

- **NFV Orchestrator (NFVO):** is in charge of the management and orchestration of NFVI and software resources, and realizing network services on NFVI.
- **VNF Manager (VNFM):** is responsible for VNF lifecycle management and operations, such as instantiation, query, scaling, update, and termination.

- **Virtualized Infrastructure Manager (VIM):** comprises the functionalities that are used to manage and control the interaction of a VNF with computing, storage, and network resources under its authority, as well as their virtualization.
- **Service, VNF and Infrastructure Description:** this data-set provides information regarding the VNF deployment template, service-related information, and NFVI information models, which are used internally within NFV-MANO.
- **Operations Support Systems and Business Support Systems (OSS/BSS):** is responsible for coordinating with the traditional network system, such as OSS and BSS to ensure the NFV-MANO, NFVI and functions running on legacy equipment with pre-defined communications interfaces. This functional element is defined in Section 2.2.

2.1.4.2 Software-Defined Network

Open Networking Foundation (ONF) is a non-profit operator-led consortium that specified and standardized SDN [43]. The architecture of SDN is specified in the ONF specifications TR-502 [44] and TR-521 [45]. SDN consists of the separation between CP (i.e., control logic) and DP (i.e., data forwarding equipment) through the definition of well-defined Application Programming Interfaces (API), where the network intelligence and state are logically centralized, gaining a global view of the entire network.

The shifting of control into manageable computing nodes, allows underlying network infrastructure to be abstracted for network services and applications [44].

The three architectural principles that support SDN are identified below and detailed in ONF TR-521 [45]:

- Decoupling of traffic forwarding and processing from control;
- Logically centralized control;
- Programmability of network services.

Figure 2.8 depicts the reference architectural framework that encompasses three layers and a management area, identified in [44] and described below:

- **Application Plane:** consists of one or more applications, each of which has unique control over a set of resources exposed by one or more SDN controller, communicating through the Application-Controller Plane Interface (A-CPI);
- **Controller Plane:** comprises a set of SDN controllers, each of which has exclusive control over a set of resources exposed by one or more network elements in the Data Plane, translating also the applications demands to have dynamic and granular control over the network resources;

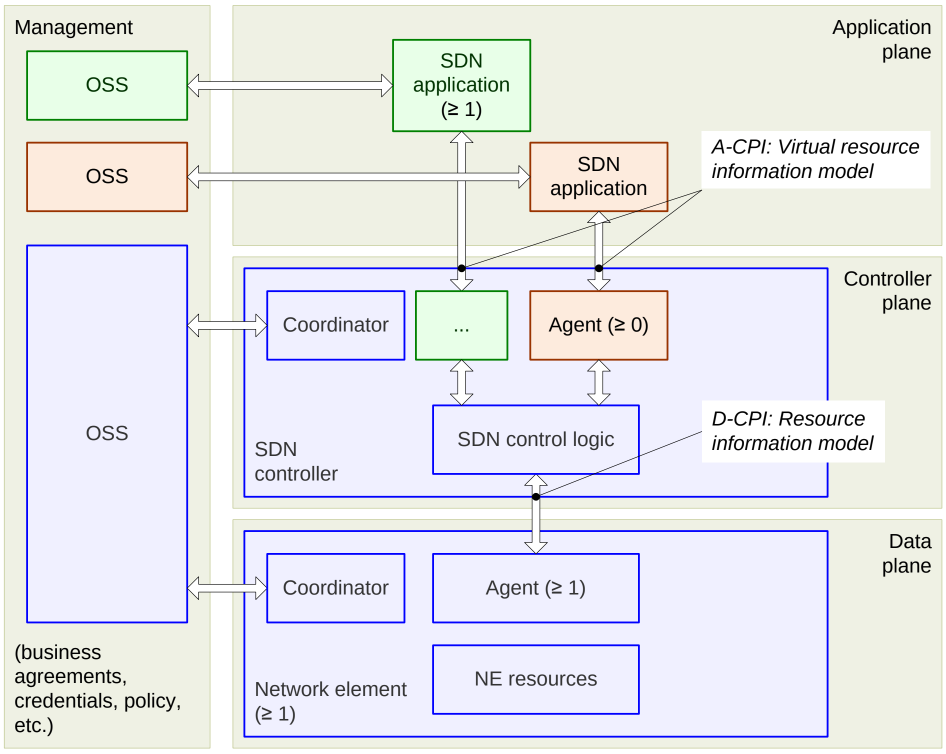


Figure 2.8: SDN reference architecture [44].

- **Data Plane:** contains the network elements, one or more, which include their functionalities and capabilities to the Controller Plane, via the Data-Controller Plane Interface (D-CPI);

- **Management:** provides managers to each application, SDN controller and network element through a functional interface to a manager, allowing for resource allocation at each layer, setting the initial configuration of the Data Plane, policy definitions at the Controller Layer or Service Level Agreements (SLA) at the Application Layer.

2.1.4.3 Relationship between NFV and SDN

NFV and SDN are complementary schemes but increasingly co-dependent. While the former offers the capability to manage and orchestrate the virtualization of resources for the provisioning of NF and their composition into higher-layer network services, the latter provides the means to dynamically control the network and the provisioning of Network-as-a-Service (NaaS).

NFV goals can be achieved using non-SDN mechanisms, relying on the techniques currently in use in many data centers. But approaches relying on the separation of the CP and DP as proposed by SDN can enhance performance, facilitate operations, maintenance procedures and simplify compatibility with existing deployments.

NFV is able to support SDN by providing the infrastructure upon which the SDN software can be run. Furthermore, NFV aligns tightly with the SDN objectives to use commodity servers and switches. So, either can be used alone, but the two can be combined to obtain greater benefits. Therefore, NFV and SDN form a mutual supplementary relation in implementing network virtualization [40, 41]. The relationship between these two paradigms is characterized in documents from ETSI in [46] and from ONF in [47].

2.1.4.4 Telco Cloud

The virtualization of networks associated with the flexibility of NFV and the programmability of SDN leverages the way networks are deployed to the Telco Cloud network infrastructure, which operations and network services are made to be Cloud-native [48].

The transformation of the telecommunications industry landscape is changing, with 4G, a Cloud-native core architecture is becoming the preferred option, but with 5G a Cloud-native design is a requirement. To meet the scalability, the flexibility, and the performance to cost-effectively deliver 5G services, a Cloud-native framework able to integrate all VNFs is required. This would enable operators to create, contract for, or require as a condition of use, a standardized adapter that would expose all control and management APIs and data in a common way.

Only by redesigning the software architecture and core functions using Cloud-native design principles and IT web-based development and methodologies, the CSP can gain the necessary agility to rapidly deliver new services and reduce their time-to-market. 5G-PPP provides first insights and trigger discussions about 5G Cloud-native design in [49] and several use cases are presented in [50].

2.1.4.5 Multi-access Edge Computing

MEC is also one of the key pillars for meeting the demanding KPI of 5G. It offers a Cloud-based Information Technology (IT) service environment and Cloud-computing capabilities at the edge of an access network, which contains one or more types of access technology, and is located close to its users. This environment is characterized by ultra-low latency and high bandwidth as well as real-time access to radio network information that can be leveraged by applications as software-only entities that operate on top of a virtualization infrastructure.

MEC has a similar approach to NFV, because it uses a virtualization platform for running applications at the mobile edge network. The main purpose of this approach is to improve the end-users service experience, reduce the load on the transport infrastructure and minimize any possible failures [51, 52].

ETSI ISG MEC specifies the identified requirements and usage scenarios in [51] and also in white papers [53] and [54]. The MEC framework and reference architecture are defined in [55].

2.2 Network Orchestration Overview

The architecture of telecommunications operators is logically organized in different layers with the bottom layers reserved for delivering services and the upper layers deal with products and marketing domains. The main reference for most CSP is the TeleManagement Forum[1] Frameworx [56], which provides a complete blueprint for the operational aspects of any digital services business, also a number of best practices and standards targeted towards architecting standardized solutions [57].

Usually, it is divided into two systems: the Operations Support Systems (OSS) that comprises all components involved in resource and service management, collaborating to provide different functionalities, such as service provisioning, monitoring and assurance, billing, and customer care; and the Business Support Systems (BSS) that includes all the components concerning the CSP business model and all the interfaces with the customer [57]. The main focus in this Section is the orchestration domain and only takes into account the OSS.

As stated in [57], "the word orchestration in the telecommunications industry comprises the coordinated execution of workflows to complete a specific objective, consisting of operations on top of services and resources which may be contained in several domains."

2.2.1 End-to-End Orchestration

The pre-condition to achieve the end-to-end vision is to enable multi-domain orchestration, which consists of the orchestration of different infrastructure domains belonging to different operators [58]. An end-to-end orchestrator is responsible for infrastructure mediation, enabling the complete lifecycle management of all domains and services under the administration of network operators [57]. This concept is outlined in Figure 2.9 as a combination of orchestration on the ETSI NFV scope, earlier mentioned in Section 2.1.4.1, on the SDN scope, discussed in Section 2.1.4.2, and on the Legacy Network Controllers scope for non-virtualized and specific technologic domains [57]. A full analysis of end-to-end multi-domain management and orchestration frameworks is realized in [58].

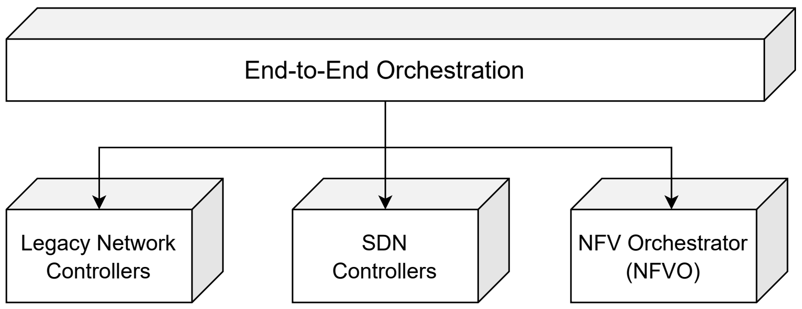


Figure 2.9: End-to-end orchestration (adapted from [57]).

[1]TM Forum is a telecommunications SDO focused on services.

2.2.2 NFV Management and Orchestration

A key goal for service orchestration is to minimize human intervention during the deployment and management of network services. Many use cases of NFV or SDN require an orchestration system that coordinates multiple vendors of hardware and software. There are many techniques for orchestration [59]. An analysis of the architecture of multiple Management and Orchestration (MANO) systems is presented in [2].

The architecture defined by ETSI NFV is commonly adopted and provides a stable base to MANO systems [60]. As stated in subsection 2.1.4.1, ETSI NFV-MANO is composed of VIM, VNFM, and NFVO. This framework is in charge of the provisioning of VNF and related operations, such as the configuration of the VNF and the infrastructure where these functions operate on, commonly in data centers. Although, its scope does not include the management of end-to-end services [61]. The ETSI specification GS NFV-MAN 001 [61] describes the functional blocks and all reference points of this architectural framework. In [2], 5G-PPP summarizes the developments in the MANO architectures by ETSI and 3GPP. Finally, the NGMN presents in [62] the key requirements and high-level architecture principles of network and service management including orchestration for 5G.

2.2.3 Advances in Orchestration Mechanisms

2.2.3.1 Closed-Loop Control

Current OSS models have static decision-making rules, and when taken the challenge of controlling the operations of an SDN and NFV virtualization enabled network, such as 5G, the lack of more granular control over the decision taken place in these systems, will lead to inefficiencies [57]. An important mechanism for the next-generation of OSS is the adoption of closed-loop control that is based on the OODA loop ("Observe, Orient, Decide, Act"). OODA loop is a methodology for the learning, growth, and thrives in changing environments, that was developed by the military strategist John Boyd during the analysis of combat scenarios as defined in [63].

The use of the OODA allows for flexibility in the adoption of a variety of new automatizations, aware of its operating environment and able to respond dynamically to observed changes, characteristics which capitalize the efficiency in the operations over the operators' Cloud-based infrastructure, and in the fulfillment of multiple and dynamic 5G requirements. It also allows for the evolution in the Self-Organized Networks (SON) way, aiming at making service and resource management more autonomous and more automatic, allowing for the reduction of the cost of the ever more complex and dynamic managing networks [64].

The OODA loop is a vital point in the process of making network management more autonomous. An interpretation of this context is provided in Figure 2.10, which involves monitoring (data collection), data analysis, decision making, and orchestration (decision enforcement) towards more intelligent management.

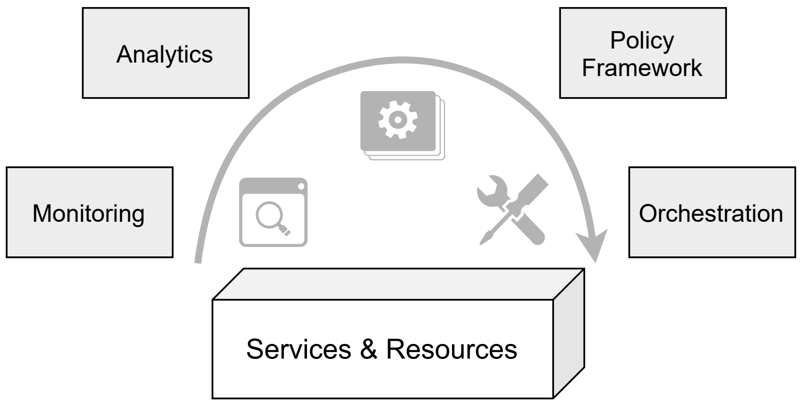


Figure 2.10: Closed-loop orchestration in network management (adapted from [57]).

2.2.3.2 Policy-Based Network Management

The Policy-Based Network Management (PBNM) paradigm defines the usage of policy rules to manage one or more entities. Each policy rule is composed of a set of conditions and a corresponding set of actions. The condition defines when the policy rule is applicable.

This paradigm was conceptualized as a set of mechanisms that can be used to "fine-tune" different network services. A PBNM-based system provides a set of mechanisms that can be used to condition traffic flowing through the network. These systems also have the ability to define a complex set of mechanisms that can be used to implement a predefined service [65].

Policy-based systems are a promising solution for implementing many forms of large-scale, adaptive systems that dynamically change their behavior in response to changes in the environment or to changing application requirements. This can be achieved by modifying the policy rules interpreted by distributed entities, without recoding or stopping the system. This aspect of policy-based management permits the system to adapt to evolutionary changes and new application requirements [66, 67].

2.3 Network Service Orchestration

There are several software artifacts related to orchestration covering from a single Cloud environment up to more complex scenarios involving multi-domain orchestration. They are the result of open-source initiatives, commercial vendors, or research projects.

Open-source approaches significantly accelerate consensus that forms a basis for an ecosystem of solutions and makes it possible to create a single unified orchestration abstraction [68]. A NSO refers to a software solution that allows network operators to configure and automate multiple network elements as per a given service definition and workflow [69]. NSO solutions need to perform management tasks, such as remote device configuration, monitoring, and fault management [68].

These solutions, help customers increase revenue by designing and deploying new services so much faster, i.e., the time-to-market is reduced from months to a few hours [69]. NSO avoids technology and vendor lock-in for investment protection. It allows easy integration and automation with customers is self-service portals, third-party Web applications, OSS/BSS via REST APIs and has no hard-coded parameters for the rollout of new device types and new services [69].

The most challenging issue in this environment is deploying and organizing in large-scale and multi-domain infrastructure. The combination of Cloud computing, NFV, and SDN can achieve the best operation. In order to realize the full potential of these technologies, operators need to use open-source developments [59]. The following subsections presents two NSO solutions as well as a summary of open-source NSO implementations.

2.3.1 Open Network Automation Platform

ONAP is an open-source software project of the Linux Foundation, which provides a comprehensive platform for design, creation, orchestration, management and automation of network and edge computing services, that will enable network operators, Cloud providers, and enterprises to rapidly automate and support complete lifecycle management of a variety of network services [70, 71]. It offers a unified framework that allows vendor-agnostic and policy-driven service design, implementation, KPI monitoring and lifecycle management enabling massive scale automation capabilities for both physical and virtual network elements [72]. The purpose is to operate a common platform that will be able to provide efficient, end-to-end infrastructure management [59].

From a high-level perspective, the architectural framework of ONAP, illustrated in Figure 2.11, is composed of two main domains with numerous subsystems separated inside of them: the Design-Time and the Run-Time environment [70].

The Design-Time is used as a development environment that allows the creation of network services with a declarative modeling language, which makes it possible to specify the requirements and functionalities of each service. It allows the modeling of resources, services, products and their management, and control functions. This is achieved by using a common set of specifications and policies for controlling service behavior and process execution, which facilitates the reuse of models and improves efficiency. It includes the Service Design and Creation (SDC) module that is used as a visual tool for designing and modeling assets used in all ONAP components [70].

The Run-Time environment is composed of several software frameworks with MANO functionalities that executes policies and rules prepared in the Design-Time environment to handle tasks, such as analytics, policy, monitoring, service orchestration for end-to-end service automation. This allows for the distribution of policy enforcement and templates among various ONAP modules, such as the Service Orchestrator (SO), Data Collection, Analytics and Events (DCAE), Controllers, Active and Available Inventory (AAI) [70].

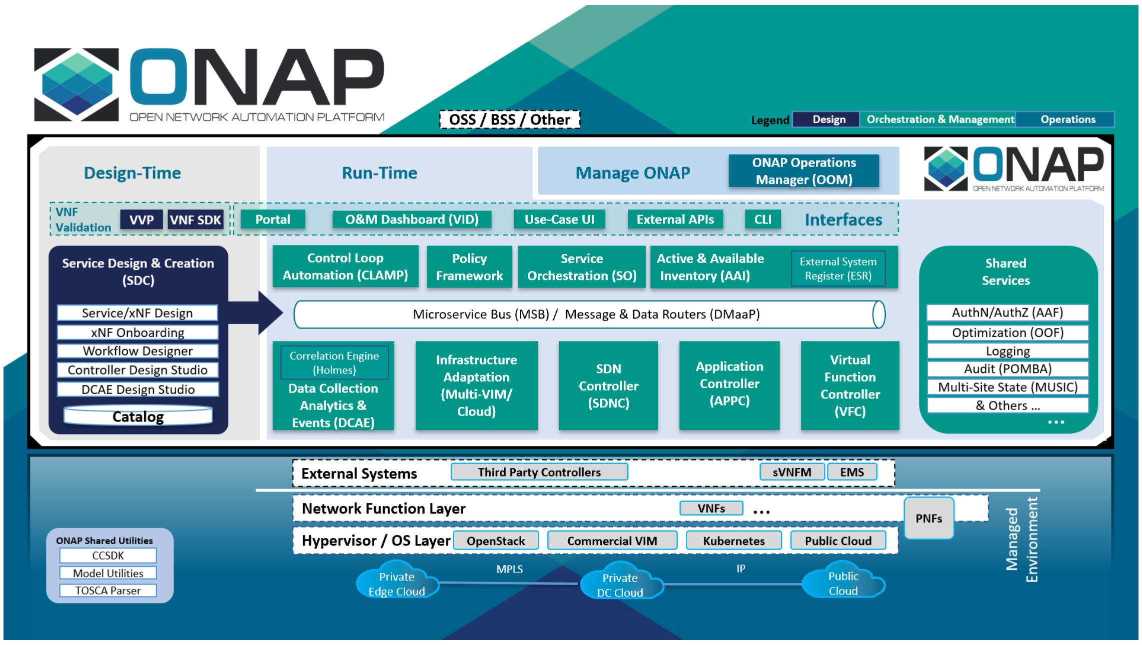


Figure 2.11: ONAP architecture overview [73].

ONAP provides the solution to the increasing demand for a proactive response to network events and service lifecycle management through a Closed Loop Automation Management Platform (CLAMP).

The ONAP deployment methodology needs to be flexible, allowing different scenarios and purposes for various operator environments. In this sense, it is possible to select only the necessary ONAP components to integrate into other systems. The platform needs to be highly reliable, scalable, secure, and easy to manage. To achieve all these goals, ONAP has designed as a microservices-based system, with all components released as Docker containers following best practice building rules to optimize their image size [70].

The ONAP community provides some reference designs, i.e., blueprints for 5G in specified use cases exploiting the combination of NFV and SDN technologies. Further information about the ONAP 5G blueprint can be found in [70] and [74].

2.3.2 Open Source Management and Orchestration

ETSI Open Source MANO (OSM) as the name implies is an open-source project conducted by ETSI to develop an NFV-MANO stack that meets the requirements of production NFV networks and is aligned with ETSI NFV Information Models and APIs. OSM enables increased multi-vendor interoperability versus traditional standardization models and proprietary tools [75, 76].

The main goal of ETSI OSM is the development of a community-driven production-quality end-to-end NSO for telco services, capable of modeling and automating real telco-grade services, with all the intrinsic complexity of production environments [77].

OSM provides the capability of realizing one of the main promises derived from NFV and the dynamic capabilities that it brings: creating networks on-demand known as NaaS for either their direct exploitation by the service provider or for their potential commercialization to third parties [77].

In that sense, OSM works as a NSO solution, intended to provide the capability of creating network services on-demand and returning a service object ID that can be used later as a handler to control the whole lifecycle and operations of the network service [77].

In OSM there are two types of NaaS service objects that OSM is able to provide on-demand to support the NaaS capability: the Network Service (NS) and the Network Slice Instance (NSI) [77]. OSM, as manager function of a service platform, consumes services from other service platforms and controls a number of managed functions in order to create its own composite higher-level service objects. The reference architecture of service platforms and common management is explored in [78].

In summary, with the significant advancements incorporated into release five, the OSM community foresees a successful adoption of the platform as a key enabler for carrier-class NFV-MANO deployments around the world. OSM has become a reliable and viable option for NFV orchestration over production environments [79].

2.3.3 Open-Source Solutions

A number of orchestration solutions based on the ETSI MANO architecture have emerged with the objective of proposing a complete orchestration framework. The main characteristics of each open-source project are summarized in Table 2.1 and are organized as follows: leader entities, VNF definition, resource domains, NFV-MANO scope, interface management, and coverage area [68].

Table 2.1: Summary of open-source NSO implementations [68].

Solution	Leader	VNF Definition	Resource Domain				MANO			Interface Management			Multiple
			Cloud	SDN	NFV	Legacy	NFVO	VNFM	VIM	CLI	API	GUI	Domains
Cloudify	GigaSpace	TOSCA	√		√		√	√		√	√	√	
ESCAPE	FP7 UNIFY	Unify	√	√	√		√		√	√	√		√
Gohan	NTT Data	Own	√	√	√	√	√	√		√	√	√	
ONAP	Linux Foundation	HOT, TOSCA, YANG	√	√	√	√	√	√	√	√	√	√	√
Open Baton	Fraunhofer / TU Berlin	TOSCA, Own	√		√		√	√		√	√	√	
OSM	ETSI	YANG	√	√	√		√	√	√	√	√	√	
Tacker	OpenStack Foundation	HOT, TOSCA	√		√		√	√		√	√	√	
TeNOR	FP7 T-NOVA	ETSI	√	√	√		√				√	√	
X-MANO	H2020 VITAL	TOSCA			√		√				√	√	√
XOS	ON.Lab	—	√	√	√			√			√	√	√

Although the progress made by ETSI in defining architecture and interfaces, each solution uses a particular implementation and data model, which makes interoperability difficult to achieve [68].

2.4 Modeling Languages

In order to add value, orchestration must address the problem of reducing time-to-market. This means that it has to be model- and catalog-driven, to stipulates both services and device configuration using a declarative data model. This Section presents the main modeling languages for this Dissertation and its value for the orchestration process.

2.4.1 Heat Orchestration Template

The OpenStack foundation defines a module called Heat Orchestration Template (HOT), which is an orchestration engine to launch multiple composite Cloud applications based on templates in the form of text files that can be treated as code [80]. In other words, OpenStack Heat is a declarative model for orchestrating OpenStack resources and managing their lifecycle, to realize the VNF onboarding [71].

Heat provides the ability to deploy instances, volumes and other OpenStack services using YAML based templates that stands for Yet Another Markup Language (YAML). YAML is a format for expressing structured data and it allows to describe the infrastructure as a code. Each section is used to describe resources, parameters, etc., and a single template describes the configuration of an entire system. Expressing these in highly readable YAML makes it easy to manage and modify system configurations, and also the fact that they are text-based makes them easy to reuse [81]. Heat provides compatibility with the Amazon Web Services (AWS) CloudFormation template format to leverage AWS products and both an OpenStack native REST API and a CloudFormation compatible query API [80]. Further information about this modeling language can be found in [82].

2.4.2 Topology and Orchestration Specification for Cloud Applications

The international consortium of vendors and users known by the Organization for the Advancement of Structured Information Standards (OASIS) defined the Topology and Orchestration Specification for Cloud Applications (TOSCA), to enhance the portability of Cloud applications and services. The main goal is to permit the interoperable description of application and infrastructure Cloud services, the relationships between parts of the service, and the operational behavior of these services (e.g., deploy, monitoring) independent of the supplier creating the service, and any particular hosting technology or Cloud provider. TOSCA will also enable the association of that higher-level operational behavior with Cloud infrastructure management.

This standardization delineates a metamodel for specifying both the structure of a service as well as its orchestration aspects. The combination of structure and orchestration in a Service Template describes what is needed to be preserved across deployments in different environments to enable automated and interoperable deployment of Cloud services and their management throughout the full lifecycle (e.g., scaling, etc.) when the applications are ported over alternative Cloud environments [83].

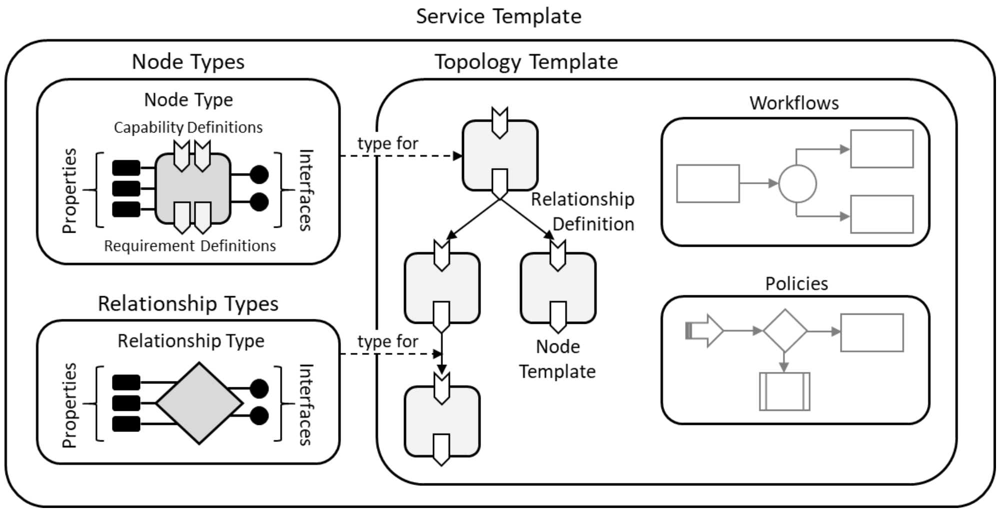


Figure 2.12: Structural elements of a TOSCA service template and their relations [84].

The Figure 2.12 illustrates the major elements of a TOSCA Service Template. Within a Service Template, a Topology Template defines the structure of a service. This template consists of Node Templates and Relationship Templates.

The Node Template specifies the required components and their properties, operations and interfaces. Each Node Template refers to a Node Type that defines the semantics of the node (e.g., properties, requirements, capabilities, interfaces, etc.).

The Relationship Template describes connectivity between components (Node Templates) by declaring the direction of the relationship (source and target) and can have additional parameters. Each Relationship Template refers to a Relationship Type that defines the semantics relationship (e.g., properties, interfaces, etc.). Node Types and Relationship Types are defined separately for reuse purposes.

Workflows aim to interpret the templates and execute appropriate actions (e.g., create, terminate) related to the management of the service is lifecycle. They have been defined as process models and they rely on the Business Process Model and Notation (BPMN), later explored in Section 2.4.3.

In order to allow automatic deployment or build of services this specification also defines Artifacts. An artifact represents the content needed to provide an implementation for an interface operation, that could be an executable (e.g., a script, an executable program), a configuration file, etc. The content of an artifact depends on its type.

Policies specify operational behavior of the service, such as QoS objectives, performance objectives, and security constraints, and allow for closed-loop automation. Further information about this modeling language can be found in [84].

2.4.3 Business Process Model and Notation

BPMN is a global standard maintained by Object Management Group (OMG) for workflows and process automation. The main objective is to provide a notation that is intuitive to business users and developers [85, 86]. Thus, BPMN creates a standardized bridge for the gap between the business process design and process implementation [86].

This specification provides a graphical notation for specifying business processes in a Business Process Diagram (BPD), which is based on traditional flowcharting techniques that are similar to activity diagrams from Unified Modeling Language (UML), tailored for creating graphical models of business process operations. It also ensures that XML (Extensible Markup Language) languages designed for the execution of business processes, such as Web Services - Business Process Execution Language (WS-BPEL), can be visualized with a business-oriented notation [87].

The modeling elements of BPMN are organized into five basic categories, such as Flow Objects, Data, Connecting Objects, Swimlanes, and Artifacts. These graphical elements can be consulted in the latest specification of OMG in [88].

In sum, BPMN is widely used in the business process modeling industry and has an easy-to-use flowchart-like notation that is independent of any particular implementation environment [85, 86].

2.5 BPM Frameworks

This Section presents two BPM tools for workflow and process automation.

2.5.1 Camunda BPM

Camunda BPM is an open-source, integrable and lightweight Java-based framework, supporting BPMN for workflow and process automation [89]. This framework is dedicated to Java developers and their typical software development infrastructure while providing Business-IT-Alignment during process design and runtime using the BPMN 2.0 standard [90]. The Camunda Consulting Team provided a manual of Camunda Best Practices available on [91].

2.5.1.1 Architecture Overview

The main components of this platform are written in Java. This framework also provides a REST API, which allows the non-Java developers to build applications connecting to a remote process engine. Camunda BPM can be used both as a standalone process engine server or embedded inside custom Java applications. The deployment of different scenarios is described in [92]. An overview of Camunda BPM architecture is provided in Figure 2.13 and the main components are described as follows [89]:

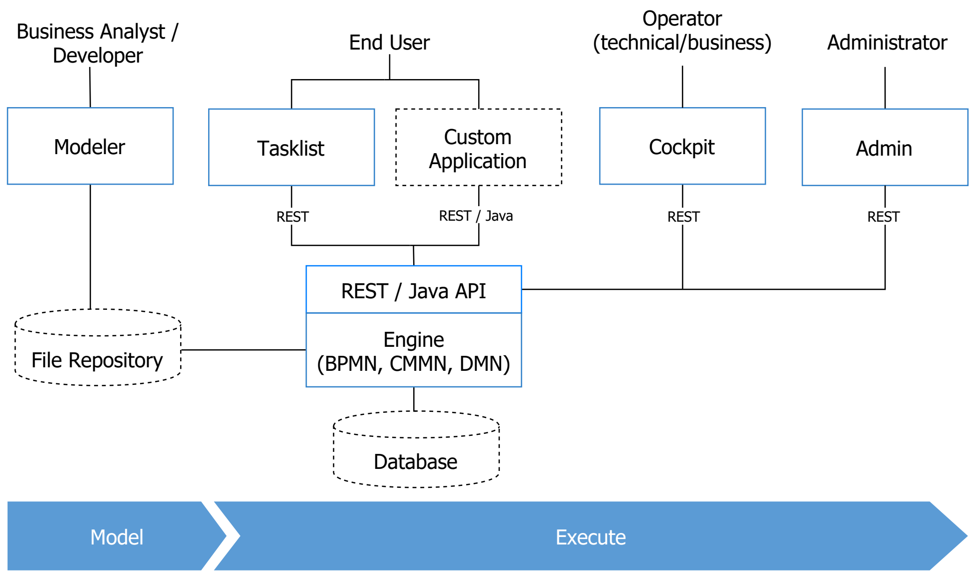


Figure 2.13: Camunda architecture overview [89].

- **Admin:** is a Web application for user management (users, groups, permissions and other configurations);
- **Cockpit:** is a Web application for process operations and monitoring, which provides mechanisms of searching, inspecting and repairing for instantiated processes;
- **Process Engine:** is a Java library responsible for the execution of BPMN 2.0 processes, it runs inside the Java Virtual Machine (JVM);
- **REST/Java API:** allows users or other applications to use the Process Engine from a remote application or a Java Script application;
- **Modeler:** is a free desktop application supported by Camunda for modeling BPMN workflows. It allows the edition of properties for technical executions.

More detailed information about these components can be found in [93].

2.5.1.2 Process Definition

A process definition defines the structure of a process and it uses BPMN 2.0 as its primary modeling language for modeling process definitions. Camunda BPM provides two BPMN 2.0 references: the Modeling Reference that introduces the fundamentals of BPMN 2.0 with examples [94]; and the Implementation Reference that comprises the implementation of the individual BPMN 2.0 constructors in Camunda BPM [95].

Camunda BPM allows the deployment of processes to the Process Engine in BPMN 2.0 XML format. The XML files are parsed and transformed into a process definition graph structure. This graph structure is executed by the Process Engine component.

2.5.2 Oracle BPM Suite

The Oracle BPM Suite is a paid software that provides a comprehensive platform with a single design-time and unified engine for process, case, rules, human tasks, forms, analytics, and integration. This software provides support for human collaboration and improves process efficiency and quality by raising utilization and throughput. It increases visibility into process performance by providing real-time analytics and simplifies compliance by offering transparent data.

The platform provides seamless round-trip between business and IT tools based on shared BPMN 2.0 metamodel leading to better business-IT collaboration. It also increases corporate agility with flexible tools [96]. More information about this software can be found at [96] and [97].

2.6 Content Delivery Network

Nowadays, video streaming occupies more than half of Internet traffic [98]. There are a number of factors which contribute to the increased usage and pressure on mobile networks through the consumption of multimedia content, such as the proliferation of devices like smartphones or tablets, the wide adoption of social media that enables quick and efficient sharing of hot topics, the access to various over-the-top media services (e.g., Netflix), and large-scale events of live streaming or the access to premium content like 8K or for AR/VR, are driving the need for even greater resource utilization of legacy infrastructure, cost efficiency, and time-to-market [99, 100]. In this context, traditional CSPs are under pressure to increase available bandwidth and to maintain competitive QoS levels.

Content Delivery Networks (CDN) have a fundamental role in the delivering of digital content to end-users and they are widely used by CSPs [100].

A CDN is an overlay network that gives more control of asset delivery while monitoring network load. It contains several servers strategically distributed in various locations, in order to cache content in close geographical proximity to the end-users, reducing response time and network congestion. When a client makes a request for content, the CDN redirects it to an optimally located mirrored server that should perform transparent and cost-effective delivery to the end-user. Location is essential for content delivery speed, e.g., if the user is far from the server where the content is stored, more time it will take for the content to reach the user, and this in turn negatively affects user experience [101, 102].

In order to reduce costs and increase elasticity, traditional CSPs have been to embrace virtualization and MEC technologies to deploy and utilize virtual Content Delivery Networks (vCDN) [103, 104].

Taking advantage of virtualization, vCDNs can be dynamically adjusted (shrunk or grown) to meet customer demand, avoiding performance, quality, reliability and availability limitations that characterize traditional CDNs [104].

The main benefits of using a vCDN are the programmability of the CDN components, like create, remove or change the location of a CDN node, adapting them to new operating conditions. And also, by enabling isolation of the virtual structures, allowing multiple instances of CDNs in a multi-provider scenario [104].

ETSI has highlighted vCDN as a major use case for NFV available in [105]. In the research article [106], an analysis of the "State of the Art, Trends, and Future Roadmap" in this area is available.

2.7 Summary

This Chapter focused on presenting a global scope about the necessary concepts for a better understanding of the various subjects covered throughout this Dissertation.

It started with an overview of the fifth-generation of mobile communications, including its standardization, usage scenarios, requirements, architecture and their key enablers, such as NFV, SDN, MEC, and Telco Cloud.

Then, the main concepts of orchestration concerning on end-to-end orchestration, NFV-MANO, and some advances in orchestration mechanisms, like closed-loop control and PBNM.

Two NSO solutions (ONAP and OSM) were introduced as well as a brief summary of the open-source NSO implementations.

Following, the major modeling languages, such as Heat Orchestration Template, TOSCA, and BPMN were described.

Later, were presented two BPM tools for workflow and process automation, both taking advantage of the BPMN standard.

Finally, considerations over the role of CDN in delivering digital content and the major advantages of virtualizing these networks were presented.

Chapter 3

Orchestration Solution

In this Chapter, the proposed orchestration solution and the use case scenario are described. In Section 3.1, the proposed architecture and description of the selected components that constitute the designed system of the NSO solution are presented. Finally, in Section 3.2, the closed-loop orchestration scenario, along with the presentation of the vCDN service and the components that enable the orchestration of this service are described.

3.1 Orchestration Platform

The orchestration platform in use is ONAP, in particular the El-Alto release. This platform, as mentioned in Section 2.3.1, is focused on the design, creation, orchestration, management and automation of network and edge computing services for CSP.

The use of ONAP represents numerous advantages, firstly because it is an open-source platform and also supports ETSI standards that promote interoperability and integration with other projects. Secondly, it allows for rapid automation and complete lifecycle management of services, which is critical for 5G and the next-generation networks.

For better understanding, this Section is organized into four subsections:

- Architecture – presents the customized architecture of this platform;
- Selected components – the used components that enable the orchestration;
- Communication – the communication buses that this platform uses;
- Interfaces – the internal and external APIs that promote interoperability and integration with other projects.

3.1.1 Architecture

This NSO solution must automate the network environment automatically. For this, a clear division between the Design-time and Run-time environment delineates three main domains: service creation, orchestration, and maintenance (operations) [73].

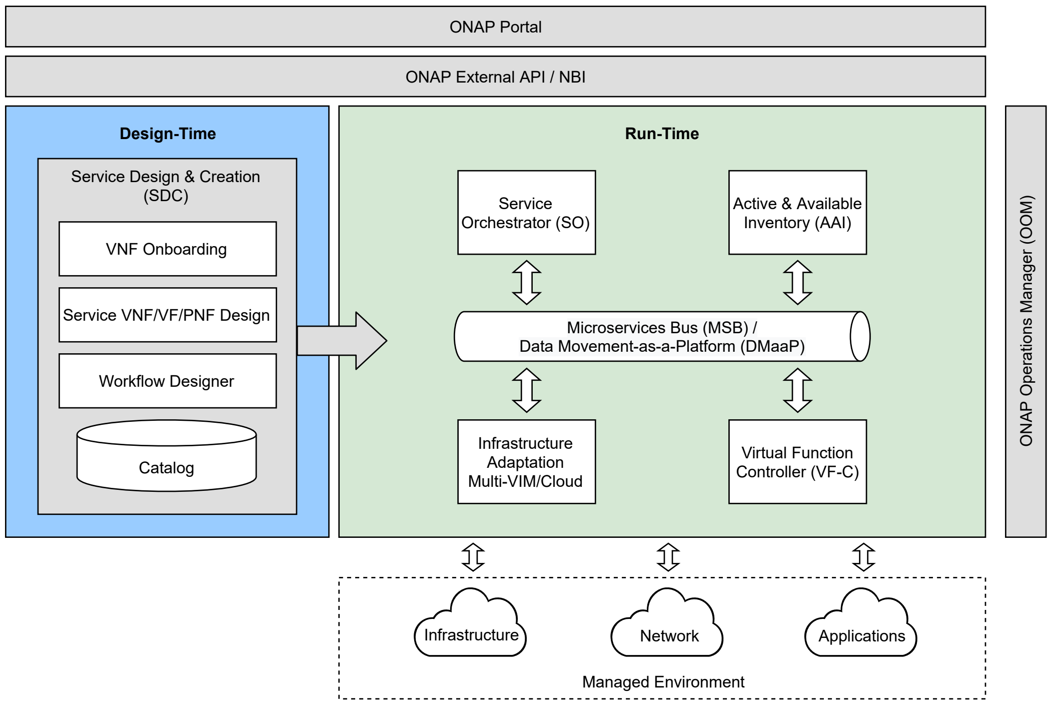

ONAP Portal
ONAP External API / NBI
Design-Time
Service Design & Creation (SDC)
VNF Onboarding
Service VNF/VF/PNF Design
Workflow Designer
Catalog
Run-Time
Service Orchestrator (SO)
Active & Available Inventory (AAI)
Microservices Bus (MSB) / Data Movement-as-a-Platform (DMaaP)
Infrastructure Adaptation Multi-VIM/Cloud
Virtual Function Controller (VF-C)
ONAP Operations Manager (OOM)
Infrastructure
Network
Applications
Managed Environment

Figure 3.1: Architecture of customized orchestration platform.

The presented solution is illustrated in Figure 3.1, which results in the subsequent ONAP modules from both environments. This is possible due to the fact that ONAP is a modular and flexible platform that allows customization. This solution only considers the necessary components and functionalities to enable service orchestration.

3.1.2 Selected Components

The selected components that are part of the proposed architecture are described in the following subsections.

3.1.2.1 Service Design and Creation

The Service Design and Creation (SDC) component [107] is a visual modeling and design tool for Design-time activities, such as VNF onboarding, workflows definition, services and policies creation, data analytic applications onboarding, and models distribution to Run-time components. It creates internal metadata that describes assets used by all ONAP components, both at Design-time and Run-time.

This component is catalog-driven, which means all the defined models are stored in a Catalog, that represents a single source of service data. The centralization of this data and the modeling of services according to predefined templates allows providers to decrease their time-to-market for new digital services.

The SDC manages the content of a Catalog, and logical assemblies of selected catalog items to establish rules, in other words, it defines how and when VNFs are realized in a target environment. A virtual assembly of specific Catalog items, together with related workflows and configuration data, defines how the deployment, activation, and lifecycle management of VNFs are accomplished.

The models defined in SDC describe asset capabilities and how they have to be managed. Inside the SDC Catalog, two levels of assets are managed:

- **Resource:** it represents a combination of one or more Virtual Function Components (VFC), along with all the information necessary to instantiate, delete, update and manage the Resource. A Resource also includes license-related information. There are three types of Resource:
  – Infrastructure, composed by the Cloud resources (e.g., storage, compute);
  – Network, constitute network connectivity functions and elements (e.g., a VNF);
  – Application, includes all features of a software application (e.g., a load balancer).

- **Service:** it is an object composed of one or more Resources. Service designers create Services from Resources, including all of the information needed to instantiate, update, delete, and manage the Service.

There are four major components in SDC:

- **Catalog** is the repository for assets at the Resource and Service levels;

- **Design Studio** is used to create, modify, and add Resource and Service definitions in the Catalog;

- **Certification Studio** is used to test new assets at all levels in order to certify them;

- **Distribution Studio** is used to deploy certified assets.

In addition, the SDC integrates a RESTful Web Services framework to expose a set of APIs to the outside used by ONAP components and also provides an Integrated Development Environment (IDE) that is accessible by designers through the ONAP portal. Moreover, it provides a role-based distinction of users (e.g., Designer, Tester, Governor), later described in Section 4.2.1.1.

After the conclusion of the service design activities, SDC offers an interface to distribute the modeled services, TOSCA artifacts and Cloud Service ARchive (CSAR[1]) files to Active and Available Inventory (AAI), Service Orchestrator (SO), SDN - Controller (SDN-C) and other ONAP Run-time components, using Data Movement-as-a-Platform (DMaaP) notifications.

[1]The SDC Service CSAR [108] is a package of artifacts, which captures the information associated with a service defined at Design-time.

3.1.2.2 Active and Available Inventory

The AAI module [109] provides real-time views of available Resources and Services and their relationships. It forms a central registry of active, available, and assigned assets, maintaining up-to-date views of the multidimensional relationships among these assets, including their relevance to different components of ONAP.

Since this component is metadata-driven, new Resources and Services can be added quickly with SDC catalog models. Data stored in its inventory is continuously updated in real-time, corresponding to the changes made within the Cloud environment, and all operations are available through a RESTful API.

AAI exploits graph data technology to store relationships between Inventory items. This allows identifying chains of dependencies between items. It displays relationships between products and the services composing them, but also between Services and their Resources. It also shows relationships among different VNF that are chained to realize an end-to-end service. The collected data can be used during service delivery, impact analysis, analysis of problems, and many other processes, thanks to providing a reliable snapshot of the current state of all resources managed by the ONAP platform.

3.1.2.3 Service Orchestrator

The SO component [110] is responsible for orchestration logic, performing real-time actions on services and automates the execution of tasks, rules and policies needed to control these services. It performs orchestration at a high-level, thanks to its end-to-end view of the network infrastructure.

To fulfill the required tasks, it interacts with four controllers, which are the Virtual Function Controller (VF-C), the Multi-VIM/Cloud, the Application Controller (APP-C) and the SDN Controller (SDNC). In addition, it communicates with SDC and AAI to obtain topologies, resource models and configurations.

The SO is invoked through events by other ONAP components or API calls from BSS applications or manually through Virtual Instantiation Deployment (VID[2]). It also takes homing[3] decisions and manages rollbacks operations in case errors occur [70].

The functional architecture of SO, illustrated in Figure 3.2, consists of the following sub-components:

- **API Handler:** handles service-level and infrastructure (VNF & network) for incoming requests through a RESTful interface to northbound clients;

- **SDC Distribution Client:** receives updated service models from SDC and populate Catalog DB;

[2]VID [111] enables users to request for the creation or deletion of services and their components.

[3]Homing is intended as the process of determining the physical or virtual resources in which workloads will be placed.

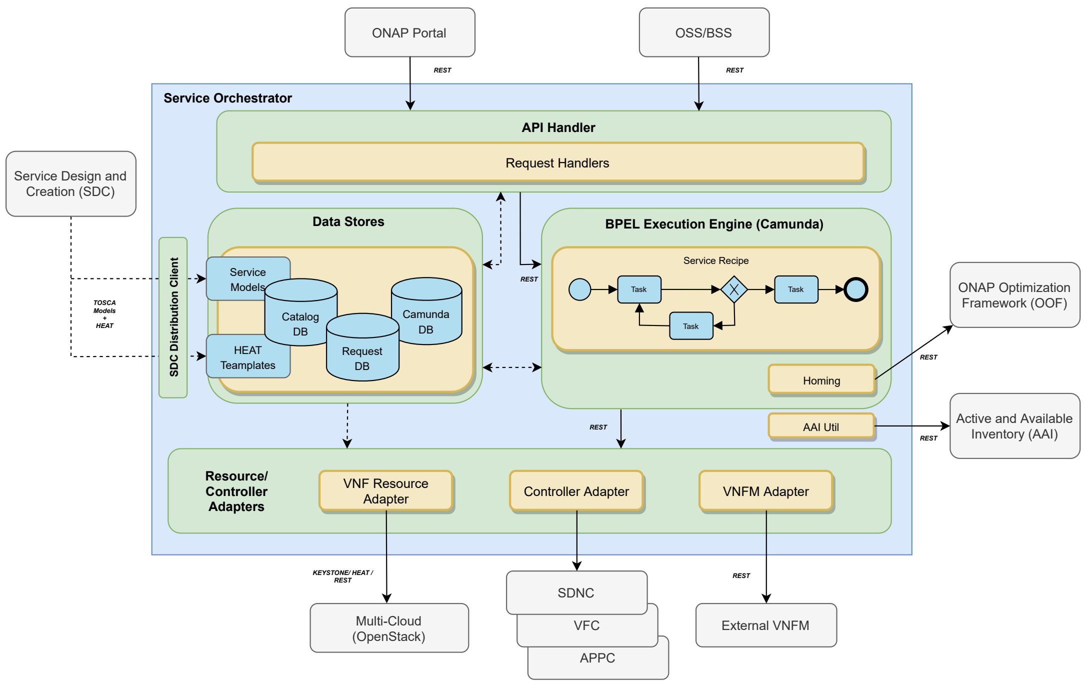


Figure 3.2: ONAP Service Orchestrator functional architecture (adapted from [112]).

- **Data Stores:** consists of three databases:
  - Catalog DB: store services and resource models, recipes, and Heat templates;
  - Request DB: tracks open and completed requests;
  - Camunda DB: maintain state for BPMN flows and supports multiple active engines.
- **BPEL Execution Engine (Camunda):** contains all business logic of service order execution and interact with AAI, SDNC, Data Stores, etc.;
- **Resource/Controller Adapters:** provides interfaces that interact with ONAP components (SDNC, VF-C, Data Stores) or external components (OpenStack, etc.);
- **SO Monitoring:** service to monitor BPMN workflows execution status.

3.1.2.4 Controllers

The Resource/Controller Adapters are the primary players in ongoing service management and they perform activities, such as service configuration and management, service migration and scaling, control loop actions, etc. [70].

The controllers act under the direction of SO or independently as triggered by events. As previously mentioned in subsection 3.1.2.3, SO interacts with four controllers, each corresponding to a different control domain. In this proposed architecture, only two are considered the VF-C and the Multi-VIM/Cloud.

The VF-C [113] is composed of two components: the NFVO (NFV Orchestrator) and the Generic VNF Manager (GVNFM), and its architecture is presented in [114].

VF-C provides reference implementation of NFVO in ETSI NFV-MANO architecture and information model as a reference. It implements Fault-management, Configuration, Accounting, Performance, and Security (FCAPS) of VNF and Network Services (NS), and lifecycle management (e.g., instantiate, scale, heal, terminate, etc.), following the ETSI NFV-MANO specifications.

The Multi-VIM/Cloud [115] handles all interactions between ONAP and all underlying VIM or Cloud platforms. It decouples the ONAP platform from the subjacent Cloud infrastructure, by providing a Cloud mediation layer supporting multiple infrastructures, e.g., OpenStack, Microsoft Azure, Kubernetes, VMware, and so on, and network backends to effectively prevent vendor lock-in.

3.1.3 Communication

To allow the exchange of information between ONAP components, the following ONAP communication buses are now presented.

3.1.3.1 Microservices Bus

The Microservices Bus (MSB) is a project [116] from ONAP that provides a resilient, reliable, and scalable communication and governance infrastructure. The main infrastructure functionalities to support ONAP microservices architecture include service discovery and registration, service routing and load balancing, timeout and retry, requests tracing and metrics collecting, etc. It also provides a service portal and service requests (e.g., logging, monitoring and tracing mechanism, etc.) to manage the RESTful APIs.

3.1.3.2 Data Movement-as-a-Platform

The DMaaP component [117] is another bus that allows communication between the platform components. It provides a premier platform for high performing and cost-effective data movement services that transports and processes data from any source to any destination. It respects the quality, security, and concurrency requirements needed to fulfill the business and customer requests, and using the appropriate format.

3.1.4 Interfaces

This subsection aims to provide an overview of ONAP APIs that promotes interoperability.

3.1.4.1 Northbound interface

ONAP embraces an architecture with well-defined APIs both within internal and across complementary projects and applications. This enables CSP and users to quickly integrate ONAP with their existing systems.

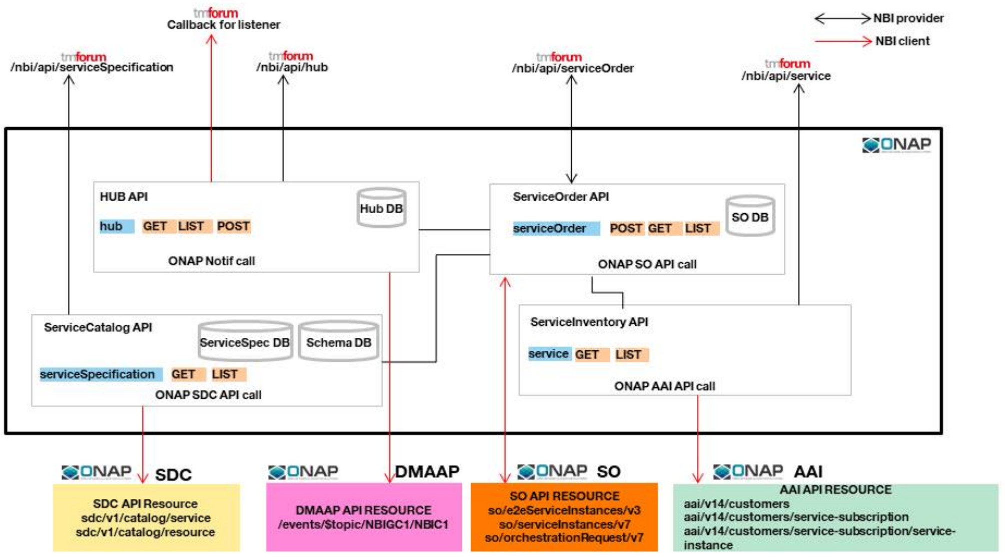


Figure 3.3: ONAP Northbound Interface architecture [119].

There are two categories of APIs in the ONAP project: the ONAP Internal API that are interfaces exposed by individual ONAP modules for exchanging information with other modules; and the ONAP External API that provides an abstracted view of the ONAP capabilities [118]. These External APIs are based on two SDO: TM Forum (TMF) and Metro Ethernet Forum (MEF[4]) [119].

The ONAP External API Framework project [120] also named ONAP Northbound Interface (NBI), brings to ONAP a set of REST APIs that can be used by external systems, such as OSS/BSS applications, to interact with other ONAP components, hiding all the complexity of these components.

A global view on ONAP NBI architecture is depicted in Figure 3.3, showing its integration with ONAP components and the available API operations. It provides access to different information stored in ONAP platform components, including SDC, AAI and SO, corresponding respectively to API REST specification TMF633 [121] (*ServiceCatalog* API), TMF638 [122] (*ServiceInventory* API) and TMF641 [123] (*ServiceOrder* API). And HUB API, also based on TMF641, is a mechanism that is implemented by subscribing to DMaaP topics and listening for events on those topics.

3.1.4.2 SouthBound Interface

For interoperability purposes, there is a need in ONAP community to highlight the importance of aligning the platform with industry standards, i.e., the ETSI NFV-MANO [124].

As explored in Section 2.2.2, the ETSI NFV-MANO framework consists of three main functional blocks: VIM, VNFM, and NFVO.

[4]MEF is a telecommunications SDO focused on Carrier Ethernet and orchestration.

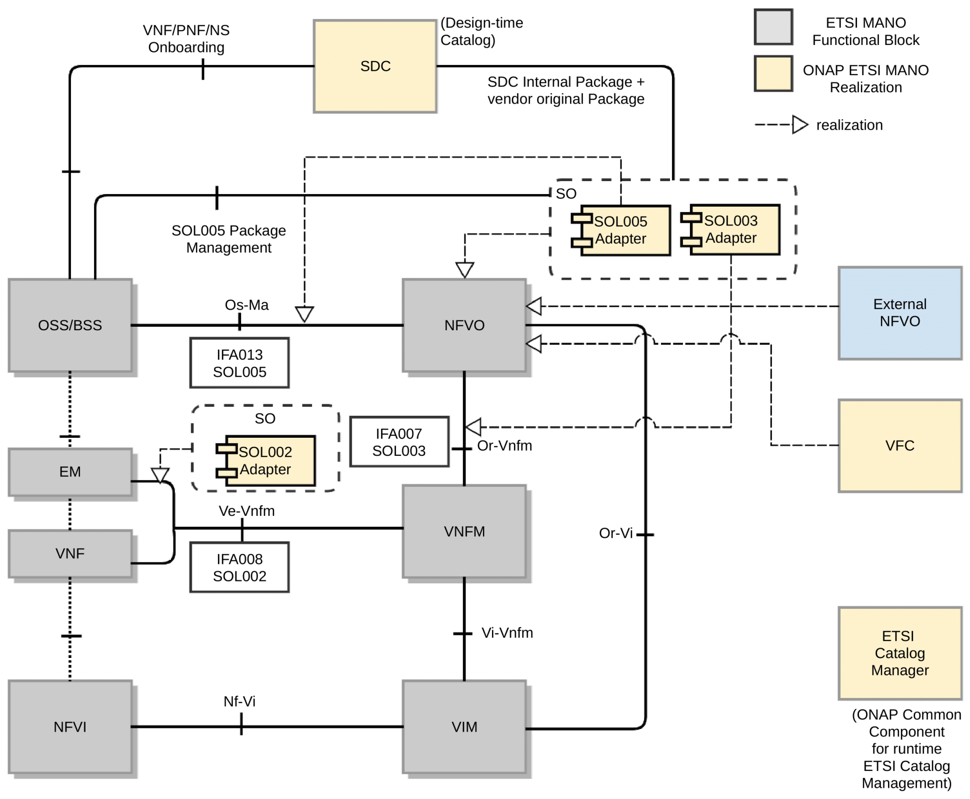


Figure 3.4: ONAP El Alto and ETSI NFV-MANO alignment (adapted from [125]).

ONAP is in line with this framework, as depicted in Figure 3.4, for building blocks, features, and interfaces [118]. As part of this alignment, ONAP supports ETSI standards for packaging, operations, and monitoring for managing VNF, Physical Network Functions (PNF) and NS [125].

From the CSP point of view, it is recommended to use standard-compliant orchestration applications and these tools need to be compatible with a subset of API and functionalities. The main interfaces for this work are highlighted below:

- ETSI NFV-SOL 005 Adapter [126] (*Os-Ma-nfvo*) used for virtual NS Lifecycle management interface, NSD Management interface, NS Performance Management interface, NS Fault Management interface and VNF Package Management interface;
- ETSI NFV-SOL 003 Adapter [127] (*Or-Vnfm*) used for VNF Lifecycle Management;
- ETSI NFV-SOL 002 Adapter [128] (*Ve-Vnfm*) used for Element Manager (EM[5]) triggered scenarios (lifecycle management, Fault, Performance, Configuration).

Furthermore, ONAP community has defined the proposals ETSI-alignment architecture, requirements and road map for future releases available in [125].

[5]EM is a component responsible for the network management functions FCAPS of a running VNF.

3.2 5G Service: vCDN

This Section intends to demonstrate the role of orchestration using a vCDN service in the scope of a closed-loop scenario to solve a network problem. First, it presents the proposed scenario, describing the resolution operation for this problem. Then, the service used to validate the orchestration platform is described, focusing on the functional architecture of the service. Finally, the orchestration role in this scenario is highlighted, presenting the components that realize the integration and interaction with this service.

3.2.1 Closed-loop Orchestration Scenario

With the aim of validating the proposed orchestration solution, a vCDN service was used in a scenario of congestion on a network link in a 5G network context, however, the 5G components are out of the scope as illustrated in Figure 3.5.

The closed-loop orchestration methodology emerged as a solution. As previously explored in Section 2.2.3.1, this methodology takes advantage of the OODA loop, making network management more autonomous involving monitoring, data analysis, decision making and orchestration.

In this context, the closed-loop orchestration is mainly formed by three components: Monitoring and Analytics, Policy Framework and SO (ONAP).

This scenario comprises a vCDN infrastructure to deliver multimedia content to the end-users. The vCDN infrastructure is composed of the Core Network, that comprehends all MANO operations of this service, and by vCDN Edge Nodes that are responsible for the delivery of multimedia content.

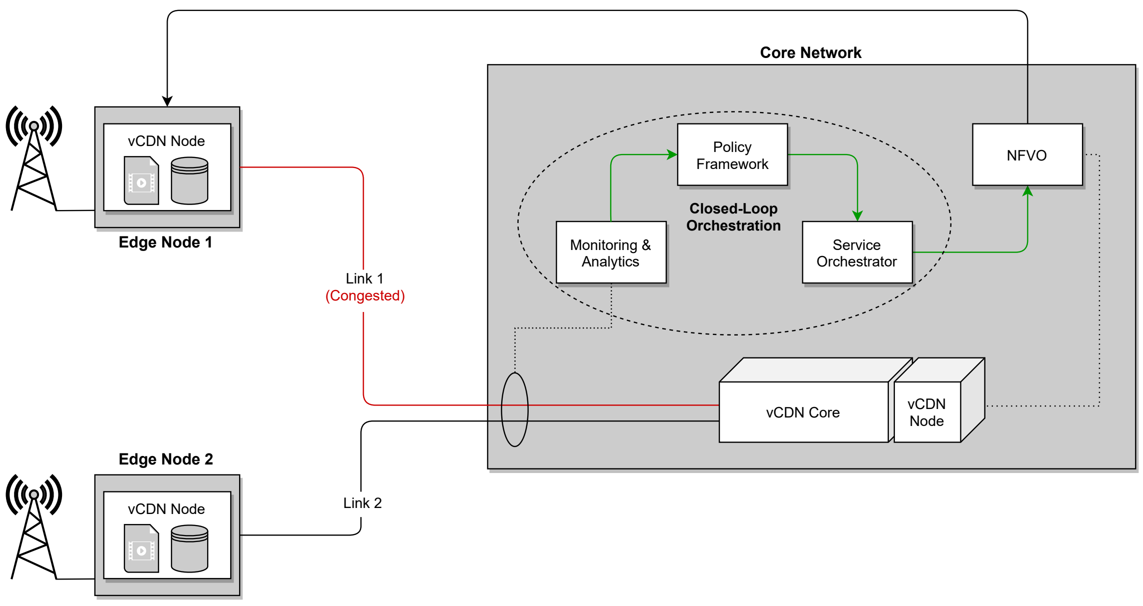


Figure 3.5: Closed-loop orchestration scenario high-level overview.

This scenario starts when there is a notification signaling the occurrence of congestion in the network link. As a precondition, the 5G network must be dimensioned so that resources can be scaled dynamically. For a better understanding and taking into account Figure 3.5, the main events are described as follows:

1. A congestion analysis system notifies that a congestion condition has been detected on Link 1, as depicted in red color;
2. The notification is received by the Policy Framework that evaluates and instructs the SO to deploy a new vCDN node on Edge Node 1, as represented in the green line;
3. The SO obtains the necessary information and realizes validations about the request;
4. The SO orders the deployment of a vCDN node to the VF-C, in particular the NFVO;
5. The NFVO orders the deployment of a vCDN node to the MEC on Edge Node 1;
6. The MEC notifies the creation of the requested node back to the NFVO;
7. The NFVO orders MEC to configure traffic redirect so that traffic flows through the vCDN node;
8. The NFVO acknowledges the creation of a vCDN node to the SO;
9. The SO acknowledges the deployment of a new vCDN node on Edge Node 1.

As a final result, the instantiation of the network elements involved has succeeded and the congestion situation was solved.

3.2.2 vCDN Service

As stated in Section 2.6, a vCDN service takes advantage of NFV and MEC technologies to cache content closer to the end-users, reducing response time and network congestion. The main goal is to improve and optimize the provisioning of multimedia content to the users, by reducing the load on the transport network and minimize any possible failures. The vCDN service in use also complies with the ETSI NFV architecture, which enables integration with ONAP.

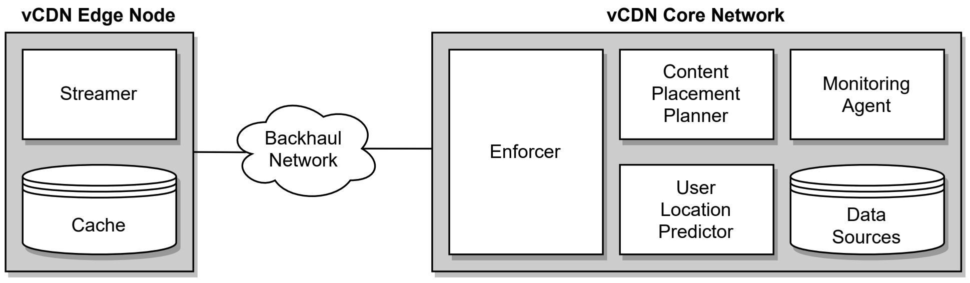


Figure 3.6: Functional architecture of vCDN service.

From a logical point of view, the functional architecture of vCDN service is depicted in Figure 3.6 and its components of Core and Edge are described in the following subsections.

3.2.2.1 vCDN Core

The vCDN Core includes the following components:

- **Monitoring Agent:** comprises mechanisms for integrating monitoring sources (e.g., User, Network), storing it on the available databases;
- **Data Sources:** are a set of databases responsible for storing relevant information in the operations of the vCDN service;
- **User Location Predictor (ULP)**: embeds various algorithms for predicting user location and/or path based on User Equipment (UE) and MEC location services;
- **Content Placement Planner (CPP):** realizes predictive caching and buffering. It uses analytical capabilities to provide recommendations to the vCDN Enforcer regarding the target caches when the content should be replicated;
- **Enforcer:** realizes the logic of the vCDN platform, by responding to northbound API requests, providing enforcement of operations recommended by the CPP, and registering in formations in the Data Sources.

3.2.2.2 vCDN Node

The vCDN Node is located in the network Edge, which means that is a component of the vCDN system that is logically distributed in the network to provide the content delivery optimizations. This component comprises two functions. The Streamer function that is responsible for the delivery of digital content to the end-users. And the Cache function that temporarily stores the received content from the vCDN Core.

3.2.3 Scale-Out Operation

As previously explored in Section 2.6, a vCDN service can be dynamically adjusted to meet customer demand, avoiding performance, quality, reliability and availability limitations.

In this context, the main focus of this Dissertation aims to explore the ability to scale-out a vCDN service in the closed-loop orchestration scenario as previously presented in Section 3.2.1. This ability is provided by NFV to scale, in a horizontal manner, composable resources instances (compute, storage, network) in order to match dynamic network traffic levels.

In the case of the vCDN service, which includes the Core part (mainly management and control components of the service) and the vCDN Node (where the video streams are located and transmitted to end-users), consists of adding additional VMs (horizontal scale) for the vCDN Edge Node where they are needed.

Since this vCDN service aims at providing ETSI NFV alignment, it would benefit from its integration into an orchestrated Cloud environment that supports the realization of these capabilities. The orchestration role of the principal ONAP components and their interactions with the vCDN service is described as follows:

- **NBI:** allows the interaction with the ONAP components in a standardized and transparent way. External OSS/BSS applications are able to order for service instantiation, by sending a *ServiceOrder* API request (later explored in Section 4.3). It also allows to query the ONAP components, like consulting information stored in AAI or list of available models in SDC;

- **SO:** this component, as previously explored in Section 3.1.2.3, performs real-time service orchestration at a high-level, thanks to its end-to-end view of the network infrastructure. This component plays an important role in the orchestration logic. In particular, BPEL Execution Engine that contains all business logic of service order execution and interacts with other ONAP components, to perform the scale-out operation (creation and instantiation) of the vCDN Edge Nodes and also their termination (in case of failure);

- **VF-C:** as stated in Section 3.1.2.4, VF-C includes two components NFVO and GVNFM. Since ONAP and vCDN service are in line with ETSI NFV-MANO reference architecture, this controller realizes the integration with this service. On one hand, it takes advantage of NFV-MANO capabilities by providing the interface *Ve-Vnfm-em* (SOL002 Adapter API) to the vCDN service through the GVNFM component, as highlighted in yellow color in Figure 3.4 of Section 3.1.4.2. On the other hand, GVNFM provides the interface *Or-Vnfm* (SOL003 Adapter API) and cooperates with the NFVO component, to take part in fulfilling the lifecycle management and FCAPS management of NS and VNFs. For last, the NFVO component communicates with SO via SOL005 Adapter API;

- **Multi-VIM/Cloud:** as a VNF resource adapter, allows the communication with the lower-level infrastructure management layer, in this case, with the OpenStack platform for the creation of new vCDN edge nodes.

3.3 Summary

This Chapter introduced the proposed orchestration solution and the use case scenario.

The orchestration platform architecture capable of enabling end-to-end service orchestration was presented, along with the description of the main components and interfaces that leverages interoperability with external systems.

For the validation of this platform, a vCDN service was used and integrated, in a scenario of congestion of a network link. To solve this problem, the closed-loop orchestration methodology was presented, by highlighting the importance and the role of orchestration in this context. Last, the functional architecture of this service in the scope of the orchestrator designer was described.

Chapter 4

Implementation

This Chapter focus on the description of the implemented solution. It starts at Section 4.1, by describing the technologies that leverage the ONAP infrastructure deployment with the considered testbed. In Section 4.2 the vCDN service modeling is described in detail. Then, in Section 4.3, considerations on ONAP NBI service instantiation requests are provided along with an example request. Lastly, Section 4.4, describes the main BPMN elements used to orchestrate the vCDN service, and the main features that the developed workflow comprise are highlighted in detail.

4.1 Infrastructure Deployment

ONAP is a Cloud-native application that consists of several services. Consequently, it requires sophisticated initial deployment as well as post-deployment management.

This Section provides an overview of the necessary concepts and technologies that ONAP utilizes in its deployment. Firstly, it introduces the ONAP Operations Manager (OOM) project and then the required software tools. Lastly, the testbed environment that this Dissertation uses is described.

4.1.1 ONAP Operations Manager

The OOM [129] is responsible for the lifecycle management and monitoring activities of ONAP components, by taking advantage of the Kubernetes container orchestrator to ensure CPU efficiency and platform deployment. OOM ensures that ONAP is easily deployable and maintainable throughout its lifecycle while using hardware resources efficiently. Additionally, it improves the ONAP platform by providing resilience enhancements and scalability to the components it manages.

In summary, Figure 4.1 aims to provide a high-level view of OOM that takes advantage of several Kubernetes built-in features, which consists of the following functionalities:

- **Deploy:** with built-in component dependency management;

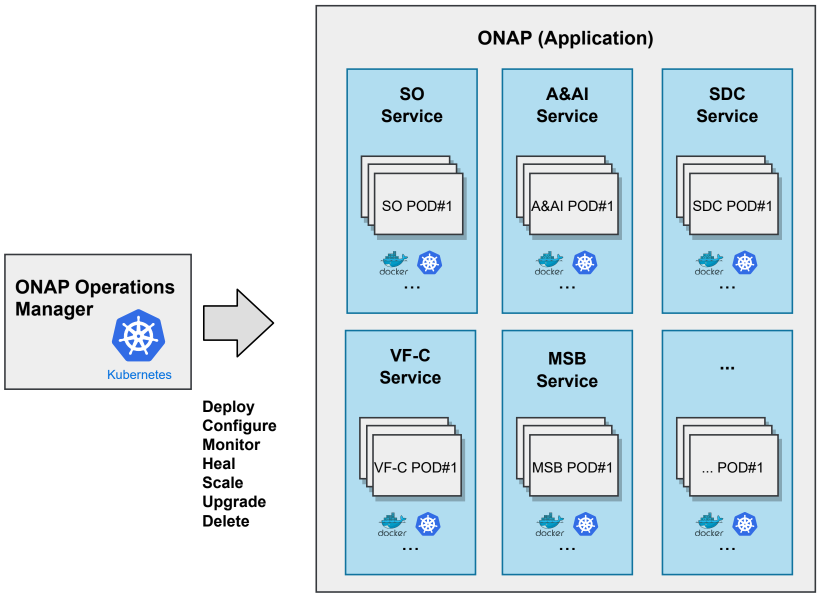


Figure 4.1: Managing ONAP with Kubernetes (adapted from [130]).

- **Configure:** unified configuration across all ONAP components;
- **Monitor:** real-time health monitoring mechanisms of its components;
- **Heal:** failed ONAP components are restarted automatically;
- **Clustering and Scaling:** cluster[1] ONAP services to enable continuous scaling;
- **Upgrade:** change out containers or configuration with little or no service impact;
- **Delete:** cleanup individual containers or entire deployments.

Furthermore, it supports a wide variety of private and public Cloud infrastructures, to suit operators' individual requirements.

4.1.2 Required Tools

The main benefits of using OOM to deliver ONAP are lifecycle management, deployment speed, hardware efficiency, and Cloud provider flexibility. In order to achieve these benefits, ONAP follows microservices-oriented architecture to be scalable, reliable, and maintainable. This approach aims to break large software projects into smaller, independent, and loosely coupled modules, which individual modules are responsible for highly defined and discrete tasks, and communicate with other modules through accessible APIs [131].

The recommended ONAP deployment is based on Kubernetes, Docker containers and Helm installer.

[1]Kubernetes cluster is a set of node machines for running containerized applications.

4.1.3 Testbed Configuration

After understanding the necessary software tools, the total minimum hardware configuration required by a full deployment of the ONAP El Alto release was set to 224 Gigabytes (GB) of Random Access Memory (RAM), 160 GB of Hard Disk (HD) and 112 virtual Central Processing Unit (vCPU), as stated in [132].

In order to support the deployment of the necessary ONAP components proposed in Section 3.1, the configured testbed comprises the resources presented in Table 4.1. The installation of ONAP was possible by using the OOM on a Kubernetes cluster in an OpenStack environment. In this case, ONAP integration with OpenStack was used as a private Cloud to manage the infrastructure.

Table 4.1: Testbed configuration for the deployment of ONAP.

Virtual Machines	**Resources**			
	vCPU	**RAM (GB)**	**HD (GB)**	**OS**
Kubernetes VM - Master	3	4	50	Ubuntu 18.04
Kubernetes VM - Worker1	6	34	80	Ubuntu 18.04
Kubernetes VM - Worker2	5	34	80	Ubuntu 18.04
Kubernetes VM - Worker3	5	34	80	Ubuntu 18.04
Kubernetes VMs Total	19	106	290	—

Once the VMs are available, the required software must be installed with the correct versions for the desired ONAP release. The installation steps are available in the "OOM Quick Start Guide" in [133]. After the successful deployment of a Kubernetes cluster, the OOM is ready to deploy and manage ONAP.

4.2 Service Modeling

This Section comprehends the necessary understanding of service modeling about vCDN service. Firstly, describes the main steps of the Design Process, which comprises the definition and distribution of service models to the components of the Run-time environment. Secondly, focus on the Service Design of the vCDN Service, by explaining is Design Process. For last, the orchestration plan for this service is presented.

4.2.1 Design-time Activities

As previously mentioned in Section 2.3.1, ONAP is composed of Design-time and Run-time environments. The Design-time framework is a comprehensive development environment with tools, techniques, and repositories for describing services, resources, and products.

First, this phase involves the definition of additional data structures, templates and software modules so that the service is properly distributed.

It is important to note that ONAP is an open-source framework which means that all documentation and code are available. This allows service providers to use the standard features provided by ONAP or customize the platform to achieve their goals.

The Design Process comprises different phases of onboarding, design, and distribution, which requires input from different users with multiple roles.

4.2.1.1 Users Management

To access the different ONAP modules, user management techniques are necessary to make the whole process more secure and organized from the administrator point of view, since each type of user can only perform a set of defined actions [134]. The user roles are the following:

- **Administrator:** triggers the actions to ultimately deploy virtual function module;
- **Designer:** is responsible for the design and creation of all the resources;
- **Tester:** is in charge of running the tests for the designed services and approving the ones in which the tests were successful;
- **Governor:** approves or denies the designs created by the Designer after they have been verified by the Tester;
- **Operator:** distributes the service throughout the platform.

4.2.1.2 Design Process

The Design Process aims at the creation of all the required models to instantiate and manage the services. This process is performed in a sequential and logical number of steps, each of which is assigned to a specific user role.

It follows the logic implemented in Figure 4.2, that demonstrates the different steps and tools involved in the Onboarding, Creation and Distribution phases. These steps are described as follows [135]:

1. **Resource Onboarding:** the Designer is responsible for Resource Onboarding. It onboards the Virtual Function (VF) and VNF, and also the corresponding models and artifacts required for the description of the software function, implemented in TOSCA templates, and for the operations on it. This includes the creation and onboarding of a License Model and of Vendor Software Products (VSP), which correspond to a VF or VNF.

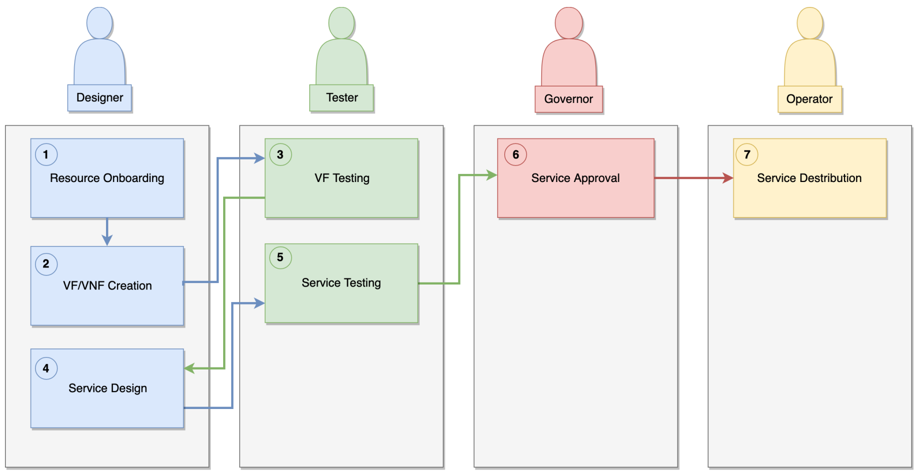


Figure 4.2: Design process in ONAP [135].

2. **VF/VNF Creation:** the VF Creation utilizes the VSP as building blocks.
3. **VF Testing:** in the Tester, the VF Testing validates the generated VF is compliance for use and adds it to the VF catalog.
4. **Service Design:** the service model and artifacts for orchestration are defined and onboarded. This step comprises the Composition of the service by adding resources and relationships among VFs, VNFs, or other Services, as well as the addition of service level artifacts. All the information characterizing the service is compressed in a TOSCA artifact and CSAR package file.
5. **Service Testing:** it is at this point that the service is tested and certified. If the tests were successful, then it is sent to the Governor for approval.
6. **Service Approval:** the Governor approves the designs and adds it to the Catalog.
7. **Service Distribution:** at this step the service is ready to be distributed to the Run-time environment. The Operator user triggers a distribution operation over the service to the target ONAP components, by disseminating the CSAR file.

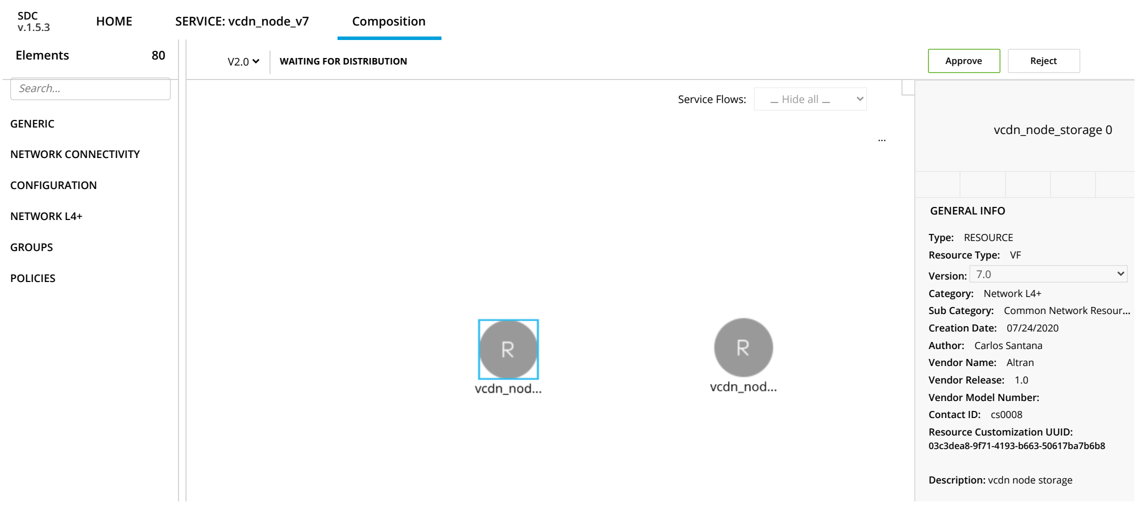
4.2.2 vCDN Service Design

The Design Process of the vCDN service (Core and Edge elements) follows the same approach described in Section 4.2.1.2. However, before the onboarding operations can be carried out, the vCDN service must be modeled in accordance with TOSCA templates.

The modeling activities here considered are only performed in the vCDN Node component for two reasons. On one hand, this component was the only one on which the orchestrator would operate in the implementation work. On the other hand, the remaining vCDN components of vCDN Core Network were already deployed in the considered orchestration target testbed of OpenStack, later described in Section 6.1.2.

The Resource Onboarding operation was already defined by describing the vCDN Node component, which is composed of two VMs, one for the Streamer function and another for the Cache function. This onboarding process takes place at the ONAP level, defining these two separate modules of the VNF Node as VF Modules through the Graphical User Interface (GUI) of SDC component, and at the TOSCA template level by describing the parameters and resources of this VMs.

After the VF Creation, which comprised the creation of the VF “vcdn_node_streamer” and the VF “vcdn_node_storage”, the Service Design step takes place including the Service Composition of the NS “vcdn_node_v7” through the GUI of SDC component, as illustrated in Figure 4.3, by adding the created VFs modules to this service.



vcdn_node_storage 0

GENERAL INFO

Type: RESOURCE
Resource Type: VF
Version: 7.0
Category: Network L4+
Sub Category: Common Network Resour...
Creation Date: 07/24/2020
Author: Carlos Santana
Vendor Name: Altran
Vendor Release: 1.0
Vendor Model Number:
Contact ID: cs0008
Resource Customization UUID:
03c3dea8-9f71-4193-b663-50617ba7b6b8

Description: vcdn node storage

Figure 4.3: Composition of vCDN node service in SDC.

At this moment, the service definition is complete. Then, the service follows the steps of service testing, approval and distribution, where the TOSCA artifacts and CSAR file will be disseminated using DMaaP notifications to the components of the Run-time environment. When a consumer application deploys a service artifact it immediately publishes a status notification. In this way, the SDC knows the state of the distribution between the different components.

4.2.3 vCDN Node Service Orchestration

As mentioned before in Section 3.2.3, the orchestration logic is implemented by the SO. This component provides to the developers several workflows to perform management and lifecycle activities.

In this context, the main component of SO that this Dissertation focus is on BPEL Execution Engine, in particular the Camunda BPM, already explored in Section 2.5.1, that allows the design and the execution of workflows.

This BPM tool enables the integration with ONAP by exposing a REST API interface and the major components are here highlighted:

- **Cockpit:** is a Web application for operation and monitoring of processes. Used with ONAP, it allows the management of workflows and its fine adjustment of the instantiated processes;
- **Tasklist:** is also a Web application that allows the inspection of the workflow tasks. It allows to and maintains navigate to task forms in order to work on the tasks and provide data input;
- **Process Engine:** is responsible for the execution of the BPMN workflows;
- **Modeler:** is the design application where BPMN workflows are modeled. It is in this component that flows are drawn and the orchestration logic is defined.

The designed solution aims at supporting two essential orchestration functionalities. The first, at the scale-out of the vCDN service, that in practical way consists of the creation and instantiation of a new vCDN Node, in case of successful operation. The second, at the handle of any possible error/failure that may occur during the operation execution, which comprises the necessary operations of termination or rollback.

The developed BPMN workflow is composed of sub-workflows and Groovy script tasks, capable to orchestrate the vCDN service including its lifecycle operations. It is presented in detail in Chapter 5, with the main macro-tasks performed identified as follows:

- Handle the scale-out request in the ONAP NBI;
- Perform the necessary validations over the request;
- Collect the necessary information about the service and available resources;
- Interacts with VF-C component to execute the scale-out operations and tracks is status;
- Concludes the workflow by informing the API Handler and ONAP NBI about the orchestration status.

After the development of the designed workflow, it is necessary to associate the workflow with the resource model in the Catalog DB of SO. When this association is completed, the API Handler is able to trigger its execution when a *ServiceOrder* API request is created.

4.3 Service Instantiation via ONAP NBI

As previously explored in Section 3.1.4.1, ONAP NBI provides an abstracted view of ONAP capabilities through the use of a TM Forum standardized API.

In this context, different methods of service instantiation are provided by ONAP. This Dissertation takes advantage of the TMF641 [123] specification (*ServiceOrder* API) for Service Orders management to perform service instantiation via ONAP NBI [136]. Through this method, the service instantiation request is converted by NBI to SO.

The *ServiceOrder* API, provides all the methods that allow to retrieve (GET), create (POST), modify (PUT), and delete (DELETE) Service Orders. An example of POST operation request body to create new service order in NBI is provided in Listing 4.1, according to the *ServiceOrder* API documentation [119].

```
 1 {
 2     "externalId": "Order_0001",
 3     "priority": "1",
 4     "description": "Altran Service E2E NBI ScaleOut",
 5     "category": "Consumer",
 6     "requestedStartDate": "",
 7     "requestedCompletionDate": "",
 8     "relatedParty": [
 9         {
10             "id": "Altran",
11             "role": "ONAPcustomer",
12             "name": "Altran"
13         }
14     ],
15     "orderItem": [
16         {
17             "id": "1",
18             "action": "add",
19             "service": {
20                 "name": "vcdn_node_service_0001",
21                 "serviceState": "active",
22                 "serviceSpecification": {
23                     "id": "ae606401-734e-405d-b601-fb726131e8d2"
24                 }
25             }
26         }
27     ]
28 }
```

Listing 4.1: Request body of ServiceOrder API.

In order to request service instance creation, the action specified for that orderItem must be “add” (a new service will be created), as shown in line 18, and the service model on which this service is based is identified by the “id” field presented in line 23, which corresponds to the “modelUUID” stored in SDC and used by the SO component in order to link the service to a specific workflow. In response to this request, the user will obtain the “serviceOrderId” value, which allows checking the service order status (completed or failed).

4.4 Workflow Elements

As stated in Section 3.1.2.3, the main components of SO are the BPEL Execution Engine (Camunda), the API Handler, the SDC Distribution Client, the Data Stores and the Resource/Controller Adapters provided to external ONAP components.

To allow the implementation of the desired vCDN orchestration operation, the focus turned to the Camunda BPM Engine, as this is the component that will execute the logic of the desired orchestration plans. Within ONAP, SO takes advantage of the Camunda BPM framework that is used for the execution of process instances.

This Section aims to provide the main workflow techniques used in this work. As explored in Section 2.5.1, Camunda BPM is an open-source framework, supporting BPMN for workflow and process automation. The Camunda Modeler provides a GUI for the creation and definition of the BPMN workflows which are executed in the Process Engine component. This framework is compliant with the BPMN 2.0 standard and, as such, defines similar BPMN modeling elements, such as Events, Activities and Gateways. The following subsections present a brief explanation of the used BPMN modeling elements.

4.4.1 Events

An Event is some action that occurs during the course of a Process. These Events affect the flow of the model and usually have a cause (trigger) or an impact (result). Events are represented by the geometric form of circles with open centers to allow internal markers to differentiate different triggers or results. There are three types of Events, based on when they affect the flow:

- **Start Event:** indicates where a particular process begins, such as receiving an order.
- **Intermediate Event:** occurs between a Start Event and an End Event. They will affect the flow of the Process, but will not start or (directly) terminate the Process. There are two main event categories: Catching and Throwing events. On one hand, when a process execution arrives at the event, the Catching Events will wait for a trigger to happen. On the other hand, the Throwing Events when a process execution arrives at the event, a trigger is fired. Examples of an Intermediate Event that uses Catch and Throw events are: Link Events for connecting two sections of a Process to avoid long Sequence Flow lines; or an Error Event, later described in Section 4.4.4.1.
- **End Event:** indicates the location where a particular process finishes, such as completing an order.

The main events that this work uses are depicted in Figure 4.4, with the Start, Intermediate (link events), and End events.

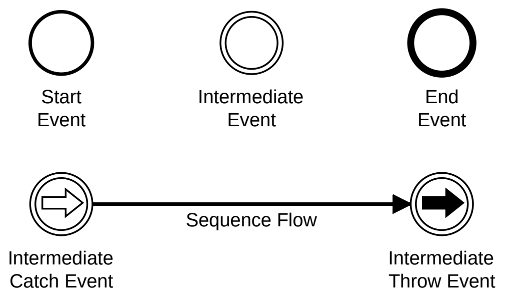


Figure 4.4: Events types.

These events are connected using Sequence Flow, which is used to connect Events, Activities, and Gateways with the goal of determining the order in which the flow objects are executed within the process.

4.4.2 Activities

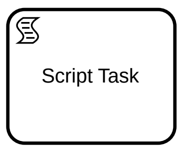
Activities represent units of tasks that require a given amount of time to be completed. Similar to Events, Activities have multiple types. The more common types of Activities that are a part of a Process Model are Task and Sub-Process, which are represented by rounded-corners rectangles.

Tasks allow for the modeling of the actual work that is being performed in the process. Tasks can be of multiple types, depending on the way they are executed. The type of task is denoted by a small symbol in the upper left part of the rectangle. The types of Tasks and Sub-Process that this work uses are described below:

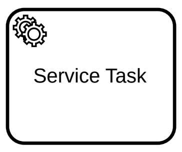
- **Script Task:** is an automated activity that runs an embedded script, its BPMN symbol is depicted in Figure 4.5a. When a process execution arrives at the Script Task, the corresponding script is executed. In this work, the developed scripts use Groovy as a scripting language. In order to handle the response body (payload) from API requests into a process variable of workflow, a JsonSlurper object was used to fetch the necessary properties of the JSON output by parsing it to text;

- **Service Task:** is used to invoke services, such as HTTP request. In this work, this type of task, depicted in Figure 4.5b, was used by taking advantage of the implementation of a *connector*, in particular the *http-connector* for the definition of inputs (URL, Method, Headers) and outputs parameters (HTTP response code and response payload);

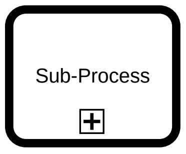
- **Sub-Process:** is used to call another BPMN workflow and is represented with a "plus" sign in the lower center of the shape as shown in Figure 4.5c. They allow to Map the input variables of the main process, in order to send to the sub-process that consists of another BPMN workflow. Then, when the sub-process is finished, it returns to the main process and also allows to Map the output variables and continue the normal behavior of the main flow.



(a) Script Task.



(b) Service Task.



(c) Sub-Process.

Figure 4.5: Activities types.

4.4.3 Gateway

A Gateway is used to control the divergence and convergence of Sequence Flow in a process. Thus, it will determine branching, forking, merging, and joining of paths. Internal markers will indicate the type of behavior control. They are commonly used when the Activities and the Events in a Process Model are not executed sequentially, and the path of the Sequence Flow needs to be controlled.

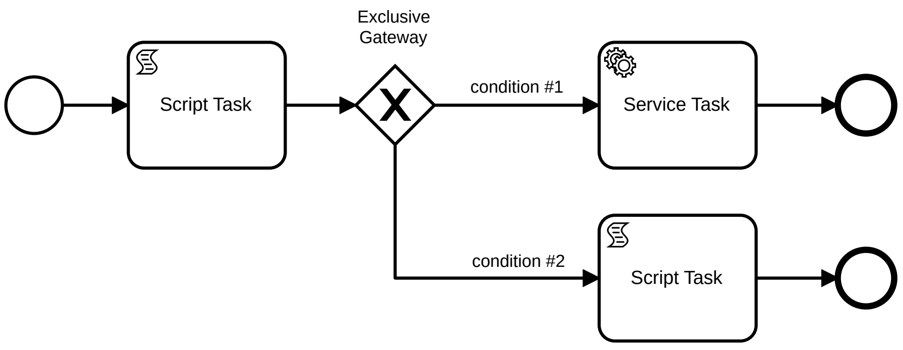


Figure 4.6: Exclusive gateway.

This work used Exclusive Gateways (an example is depicted in Figure 4.6), which are responsible for selecting one path out of multiple alternatives, based on the implemented condition.

4.4.4 Error Handling

In the developed workflow it was necessary to find error handling strategies, in order to handle any possible error, like for example an unavailable service, by catching that error event and process an error message. In this context, Error Events and mechanisms of retry were implemented.

4.4.4.1 Error Event

An Error Event is used to signal the error, i.e., they are events that are triggered by a defined error, based upon an Intermediate Event attached to the boundary of an Activity that occurs during the performance of the Process. An Error Catch Event is responsible for catching an error and is represented with a white lightning symbol, while an Error Throw Event is represented by a black lightning symbol and is in charge of trigger an error, as depicted in Figure 4.7.

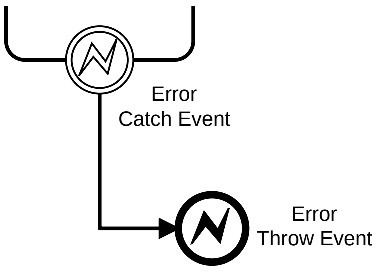
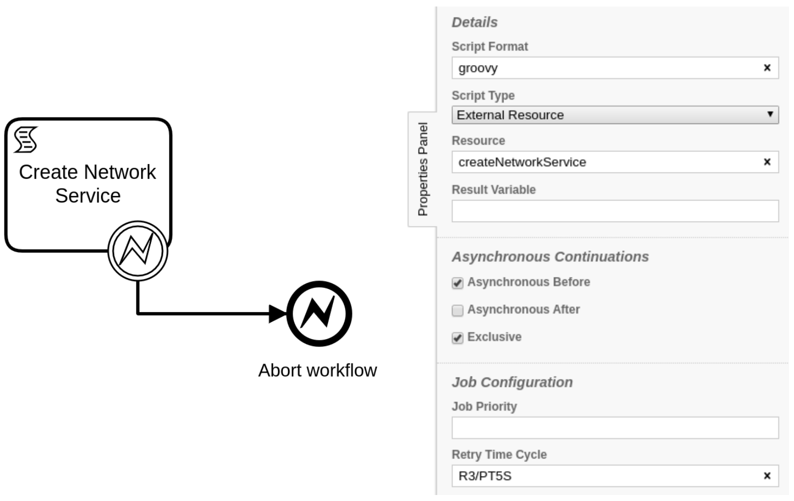


Figure 4.7: Error event.

4.4.4.2 Retry Activity

Camunda provides a retry time cycle mechanism that allows for the configuration of how often a job is retried and how long the engine should wait before trying to execute a job again. This configuration follows the International Organization for Standardization (ISO[2]) 8601 standard for repeating time intervals. An example of this configuration is depicted in Figure 4.8, by using “R3/PT5S”, which means that the maximum number of retries is 3 (R3) and the delay of retry is 5 seconds (PT5S).



Details
Script Format: groovy
Script Type: External Resource
Resource: createNetworkService
Result Variable:

Asynchronous Continuations
[✓] Asynchronous Before
[ ] Asynchronous After
[✓] Exclusive

Job Configuration
Job Priority:
Retry Time Cycle: R3/PT5S

Figure 4.8: Retry mechanism.

[2]ISO is an international standard-setting body composed of representatives from various national SDO.

Another option available is to avoid static interval by configuring the list of retry intervals separated by commas, like e.g. “PT5S,PT20S,PT40S”. In this way, it will retry 3 times and the behavior for this example would be the following:

- Job fails for the first time and will be retried in 5 seconds (PT5S);
- Job fails for the second time and will be retried in 20 seconds (PT20S);
- Job fails for the third time and will be retried in 40 seconds (PT40S).

Furthermore, as shown in Figure 4.8, this mechanism was implemented with an Intermediate Error Events (Catch and Throw), which allows error handling after exhausting the number of attempts.

4.5 Summary

This Chapter presented the various stages of the implementation of this work, from the deployment of the NSO solution to the design and orchestration of the vCDN service. And also a detailed explanation of the BPMN elements used.

The testbed environment was described along with the required tools for the deployment and management of ONAP to enable service assurance and reliability. Then, the modeling steps at the Design-time of the vCDN service was presented. As well as the design application that allows for the drawn and definition of the orchestration logic.

Following the presentation of the used method to perform service instantiation along with an example request for the execution of the workflow.

Last, the BPMN elements used in this work were covered, which include Events, Activities, Gateway, and Error Handling mechanisms, related to the Operation Design presented in the next Chapter.

Chapter 5

Workflow Deployment

This Chapter addresses the operation design by presenting the adopted methodology to orchestrate the proposed Use Case scenario and highlights the activities that perform the scale-out operation of the vCDN service.

5.1 Process Methodology

In order to handle the orchestration challenges of a vCDN service, presented and described in the previous Chapter, a three stages methodology was adopted. Each stage addresses a different level in the operation of the service:

1. **Enrichment:** comprehends the validation and preprocess of the service instantiation request to obtain the necessary information to perform the operation of scale-out;

2. **Execution:** identifies the necessary controllers that will interact with the infrastructure, resources and services to execute specific operations;

3. **Conclusion:** concludes the workflow, updating the information about the orchestration status in the inventory and handling any possible error that might occur during the operation process.

In the following subsections, more detailed information is provided at each step of the workflow, describing the relationships with the different ONAP components and their logic.

5.1.1 Enrichment Stage

The workflow is executed when a ServiceOrder request is received on ONAP NBI component. The ONAP NBI is in charge of validating the Service Order, as the API specification defined by TM Forum [123], then order processing can be continued, otherwise, an error code is returned. The service instantiation request was previously explored in Section 4.3. When this request arrives from ONAP NBI, the API Handler from SO is responsible to manage the request and parse the body to retrieve information data about service model.

The connection to the API Handler is kept open until the main process flow sends back a response. Using the model name of the resource, API Handler queries "service_recipe" table from SO Catalog DB in order to invoke the BPMN workflow associated with the service model name. Then, SO requests to Active and Available Inventory (AAI) for the creation of a new record in AAI representing a service instance to which other resource instances, belonging to the same service, will be correlated. In this way, AAI creates a hierarchical tree of relationships among instances for describing the service. To accomplish this stage, some requirements must be taken into account in its design. These arise from the following identified tasks:

- Handle of the NBI request for vCDN scaling;
- Validate if any Network Service Instance already exists in a specific Cloud Region;
- Collect information about vCDN service;
- Obtainment of available resources information.

For a comprehensive explanation, the next subsections are divided into different groups of activities.

5.1.1.1 NBI Request Handling

The workflow starts when the SO receives an API request from NBI to scale-out the vCDN service. The first task which takes place is the preprocess of the incoming request, as shown in Figure 5.1, in order to gather all necessary information about the service and saving that information into the process variables of the workflow. It also informs the AAI about the orchestration status (*assigned* or *pending*) and sends an asynchronous response to the API Handler about the status of the request.

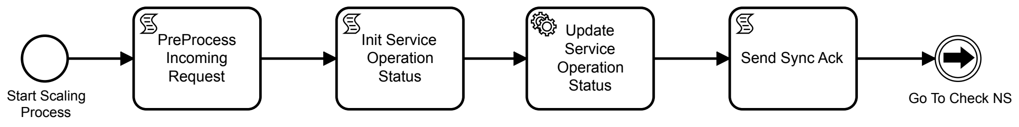

Start Scaling Process
PreProcess Incoming Request
Init Service Operation Status
Update Service Operation Status
Send Sync Ack
Go To Check NS

Figure 5.1: Handle of the NBI request for vCDN scaling.

5.1.1.2 Cloud Region Validation

The purpose of this validation aims to verify if any Service Instance already exists in the specific Cloud Region. It starts by querying AAI with the "globalCustomerId" and "serviceType" from Service Order, in a way to obtain all the Service Instances IDs, as depicted in Figure 5.2. Then, it processes the response by checking if any Service Instance was already created. If no Service Instance exists, it advances to the next task of the workflow.

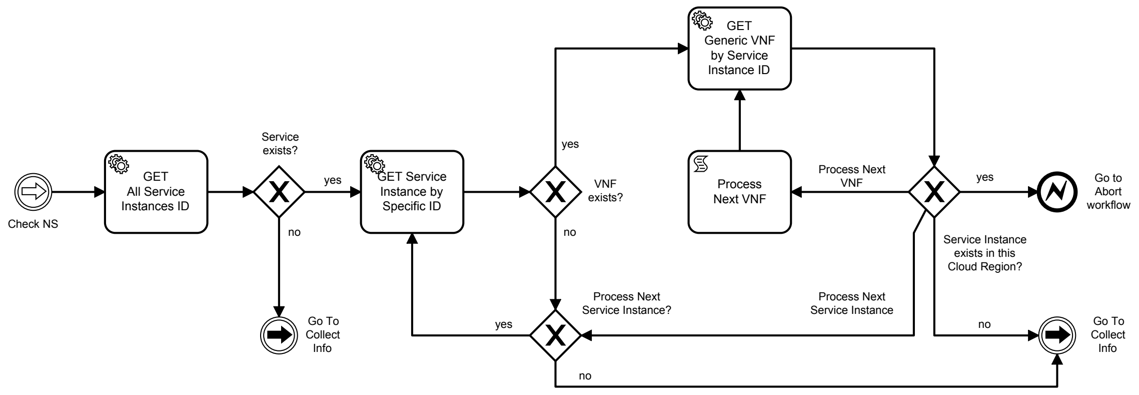

Check NS
GET All Service Instances ID
Service exists?
yes
no
Go To Collect Info
GET Service Instance by Specific ID
VNF exists?
yes
no
GET Generic VNF by Service Instance ID
Process Next VNF
Process Next VNF
yes
Go to Abort workflow
Service Instance exists in this Cloud Region?
Process Next Service Instance?
Process Next Service Instance
yes
no
no
Go To Collect Info

Figure 5.2: Cloud Region validation.

If any Service Instance exists, it obtains the necessary information to be able to compare the values of "cloudRegion" and "cloudOwner" from the NBI request with the queries made to AAI. If yes, an error message will be generated, and the workflow is aborted as a vCDN node is already in that Cloud Region. This means that Service Order to scale-out the vCDN service is not valid. Otherwise, it advances to the next task of the workflow.

5.1.1.3 Collect vCDN Service Information

After the previous validation, the BPMN workflow queries the AAI in a way to obtain all Service Instances IDs, but this time with a different Service Type, as depicted in Figure 5.3. It uses the "serviceType" from vCDN Core to obtain the Internet Protocol (IP) address. This information will be used in the creation and instantiation operations. When the IP address is obtained and saved into a process variable, the workflow advances to the next task.

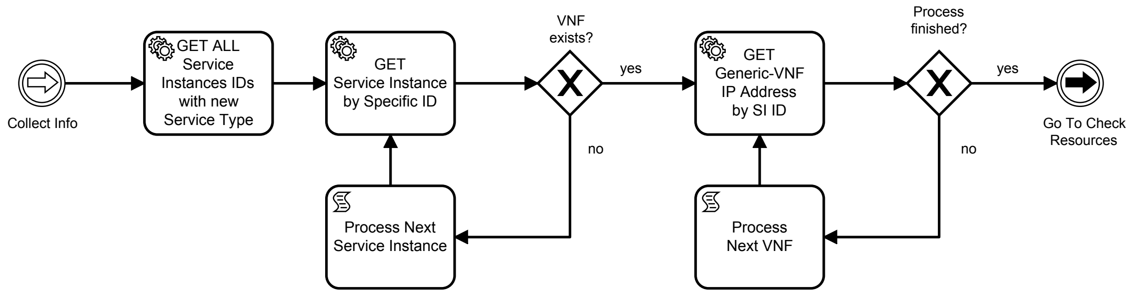

Collect Info
GET ALL Service Instances IDs with new Service Type
GET Service Instance by Specific ID
Process Next Service Instance
VNF exists?
yes
no
GET Generic-VNF IP Address by SI ID
Process Next VNF
Process finished?
yes
no
Go To Check Resources

Figure 5.3: Collect necessary information about vCDN service.

5.1.1.4 Access Required Resources

At the last operation of the enrichment stage, as depicted in Figure 5.4, the workflow accesses the necessary resources (NS and VNF) and maintains this information in process variables.

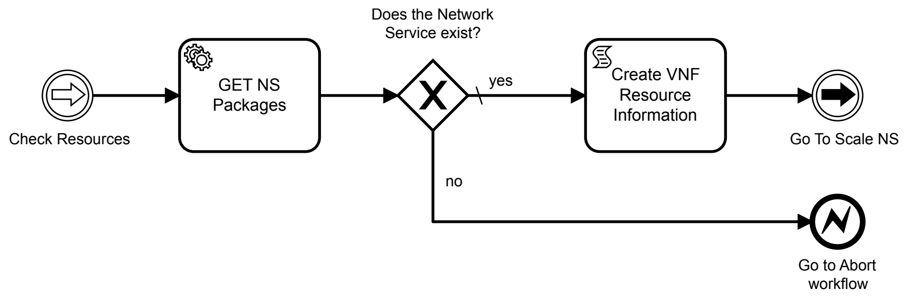


Figure 5.4: Collect information about required resources.

The SO interacts with the VF-C component by making a query, available in Appendix A.1.1, for the NS Descriptors (NSD) through NSD Management interface using ETSI NFV-SOL 005 Adapter to obtain the VNF Packages IDs ("vnfPkgIds") and store them in an Array list only if the "modelUUID" of the NBI request matches with the "id" from the query made to VF-C. If not, the NS does not exists and in this way, an error message will be generated and the workflow will be aborted.

Otherwise, it will advance to the next Script Task by querying VF-C, using the API request available in Appendix A.2.1, through the VNF Package Management interface using ETSI NFV-SOL 005 Adapter, by using each VNF ID from the previous Array list to obtain the CSAR ID ("csarId") and VNF Descriptor ID ("vnfdId"). Finally, this will result in a new array list, comprising the information of VNF Packages available on VF-C, that will be used in the Execution stage.

5.1.2 Execution Stage

This stage performs the scale-out of the vCDN service, based on the information collected in the previous stage, it realizes the operations of creation and instantiation, along with monitoring of instantiation progress. The identified tasks for this stage are the following:

- Prepare information to scale-out vCDN service;
- Creation and Instantiation requests;
- Monitoring of instantiation progress;
- Termination request;
- Monitoring of termination progress;

5.1.2.1 Prepare Information

Figure 5.5 depicts the workflow that prepares the information to send to the subflow that will perform the scale-out operation of the vCDN service. For last, the task that informs API Handler about the state of the workflow is further explained in Section 5.1.3.1.

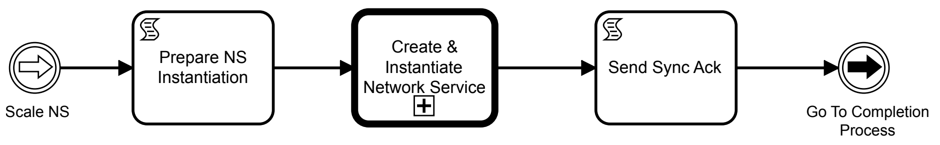


Figure 5.5: Prepare information to call subflow to scale-out vCDN service.

5.1.2.2 Creation and Instantiation Requests

At subflow, as illustrated in Figure 5.6, the workflow begins by receiving the parsed information from the main flow. Then, it realizes an API request to VF-C to create the Network Service. This API request is available in Appendix A.3.1. If the request was well succeeded it will proceed to the next Script Task. Otherwise, it will retry the request until a maximum of 3 tries with a variable time interval between every request, as previously presented in Section 4.4.4.2. In the case of after 3 unsuccessful attempts, the workflow will be finished in the End Event named "Create NS Failed" (generating an error message and updating the workflow status with *failed*). This mechanism of retry aims to safeguard and make the workflow more robust in case of service unavailable endpoint or network error at the moment the request is made.

After creating the Network Service, the instantiation request, available in the Appendix A.3.2, takes place also implementing the retry mechanism for the reliability of workflow.

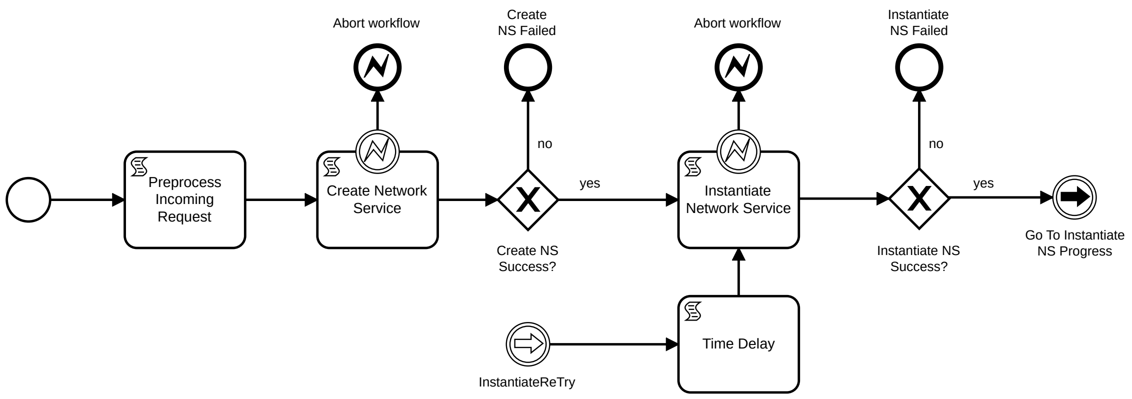


Figure 5.6: Creation and instantiation requests to scale-out vCDN service.

5.1.2.3 Monitoring of Instantiation Progress

In response to the instantiation request, a "jobId" variable is received in order to monitor the instantiation progress. Figure 5.7 illustrates the workflow for monitoring the instantiation progress. It uses the "jobId" to query the VF-C interface about the instantiation progress in the loop cycle with a Time Delay defined between every query. This API request is available in Appendix A.3.3.

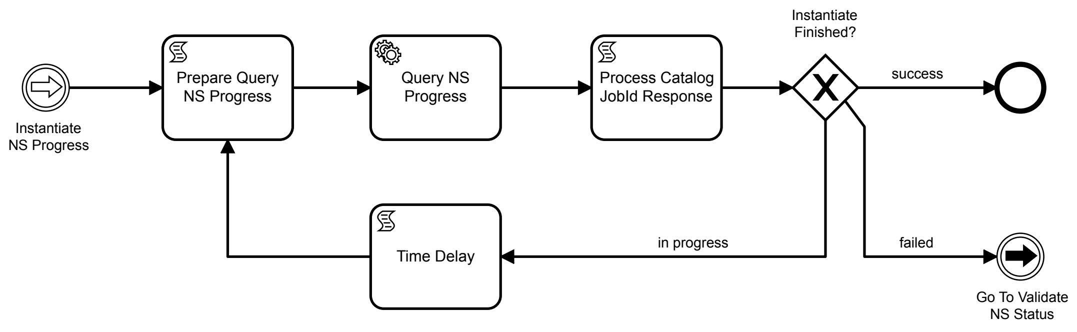


Figure 5.7: Monitoring of instantiation progress.

When the instantiation process is finished, the "jobId" status is updated from *processing* to *finished* or to *error*. If it concludes with *finished* status, it was well succeeded, thus return to the main flow and inform the API Handler, as mentioned before. If the instantiation process terminates with an *error* status, it shall retry the instantiation request, but before, it will proceed to the termination request in order to terminate the Network Service.

5.1.2.4 Network Service Validation

The operations of error handling and rollback starts at this point of workflow. Before the termination of Network Service, it is necessary to validate its status, as shown in Figure 5.8: if the network instance status is *ACTIVE* or *FAILED*, it will proceed with the termination process, this API request is available in Appendix A.3.4. If the status is *NOT_INSTANTIATED*, the instantiation process failed for some reason, i.e., did not instantiate correctly the Network Service. In this case, the workflow completion process occurs with an error message, and the *failed* status is defined.

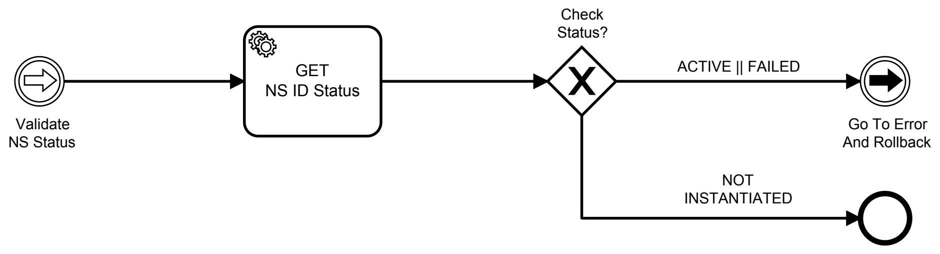


Figure 5.8: Network service validation status.

5.1.2.5 Termination Request

The termination process, as illustrated in Figure 5.9, consists of making a request to the VF-C interface ordering the termination of the desired Network Service. This API request is available in Appendix A.3.5.

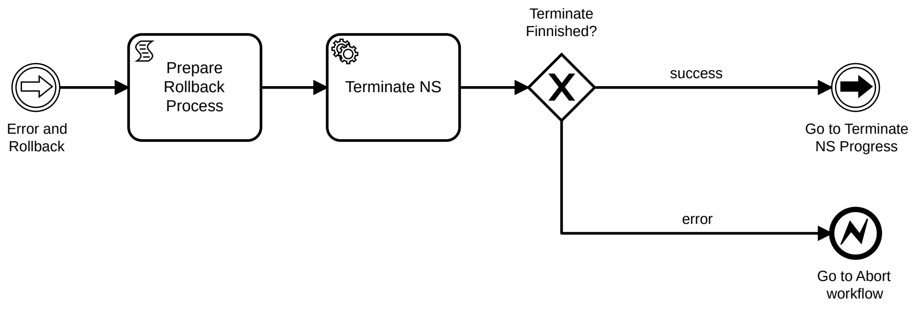


Figure 5.9: Termination request.

In response, the VF-C will return a "jobId". Thus, using this "jobId" is possible to follow the termination process, as illustrated in Figure 5.10, by querying the VF-C interface through the use of the API request available in Appendix A.3.3. When the termination request is finished, the workflow evaluates the mechanism of retry, if already tried to instantiate the Network Service until three times without success it will generate an error message and abort the workflow.

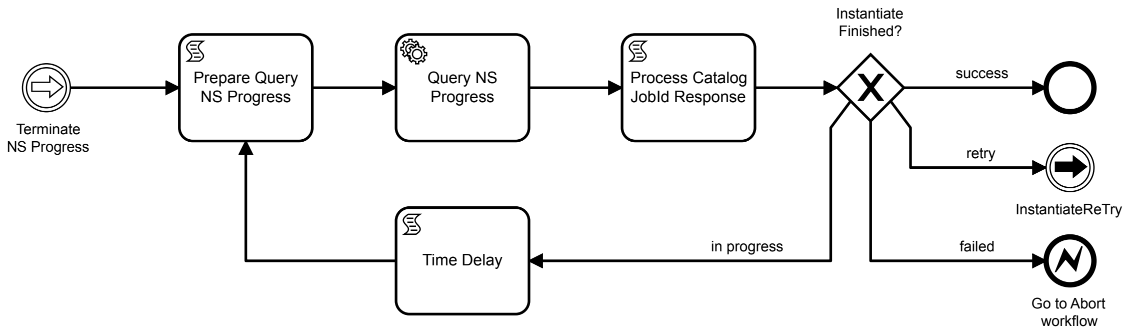


Figure 5.10: Monitoring of termination progress.

Otherwise, the termination process will be finished with success status and returns to the main flow, in that moment it will reply to the API Handler about the orchestration status.

5.1.3 Conclusion Stage

This stage comprehends all operations to conclude the workflow. The following scenarios were identified:

- Workflow conclusion;
- Workflow with error:
  - Abort workflow;
  - Handle unexpected errors.

5.1.3.1 Workflow Conclusion

The workflow conclusion is executed after the task "Send Sync Ack", as depicted in Figure 5.5, that is in charge of informing the API Handler about the status of the workflow. As depicted in Figure 5.11, the workflow concludes by returning one of two possible states: *complete* or *failed*. The obtained status will depend on the result of the instantiation process, if the Network Service was successfully instantiated the status *complete* will be set. Otherwise, an error message and the status *failed* will be generated.

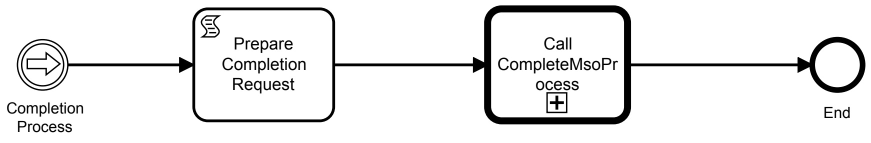


Figure 5.11: Conclusion process.

5.1.3.2 Abort Workflow

In case of aborting the workflow, due to the reasons mentioned before, such as the existence of a Network Service in the same Cloud Region as the scale-out request or when some operation over the Network Service fails, the workflow is depicted in Figure 5.12 will be executed. This workflow aims to generate an error message and the status *failed* will be defined and reported to the API Handler.

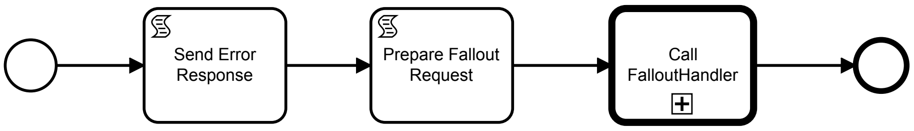


Figure 5.12: Abort workflow.

5.1.3.3 Handle Unexpected Error

For last, in case of any other errors occur, the handling of such errors is performed by the workflow depicted in Figure 5.13. This workflow also generates an error message and informs the API Handler about the *failed* status.

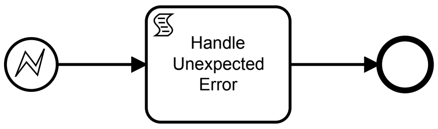


Figure 5.13: Unexpected error.

5.2 Summary

This Chapter focused on presenting the operation Design with a global scope on the methodology for performing the orchestration of vCDN service. In this context, the methodology adopted was divided into three different operational stages.

The enrichment stage carried out some important validations on the requested service and the available resources.

Then, in the execution stage comprised the main operations of the management of the lifecycle that allows the scale-out of the vCDN Edge Node.

Finally, the conclusion stage comprehended the operations of the conclusion and notification of the other components about the state of the orchestration.

Chapter 6

Obtained Results

This Chapter describes the tests performed and the achieved results, with the objective of improving the quality of the designed workflow. In particular, the execution time and the number of requests made in two specific operations of the vCDN service lifecycle management. It begins in Section 6.1, describing the study method and testbed configuration. Following, Section 6.2 presents the used approach to obtain reference values. In Section 6.3, the obtained results for each value of Time Delay defined are presented. Finally, the analysis of the results and the respective discussion are presented Section 6.4.

6.1 Study Method and Testbed

In order to improve the quality of the workflow developed, two specific and similar operations of the vCDN service were identified. In particular, the operations of monitor the instantiation and termination progress, previously described in Section 5.1.2.3 and 5.1.2.5, respectively. Both operations have a well-defined waiting time, named Time Delay, between each query made to the VF-C interface in order to obtain the "jobId" status. In this sense, the reasons that led to the selection of the Time Delay value are as follows:

- **Avoid excessive API requests:** the defined time value of Time Delay should not be short, as it is not intended to overload the VF-C interface with an excessive number of requests (e.g., in a scenario with a huge number of requests);
- **Avoid long time delays:** the defined time value of Time Delay should also not be long, preventing the workflow process from being stopped for a long period of time waiting for the opportunity to make a new query to the VF-C interface. Resulting in loss of operation time, as the SO will only execute the last query long time after the "jobId" status has already been updated by the VF-C.

In this way, when selecting a value for the Time Delay, the reasons previously presented will be taken into account, in order to find a balance between the operation execution time and also the number of requests made to the VF-C interface.

The study method and the testbed are described in the following subsections.

6.1.1 Study Method

The operations considered consist of monitoring the progress status, involving three participants in their interactions: the SO, VF-C and OpenStack, as shown in the sequence diagram of Figure 6.1.

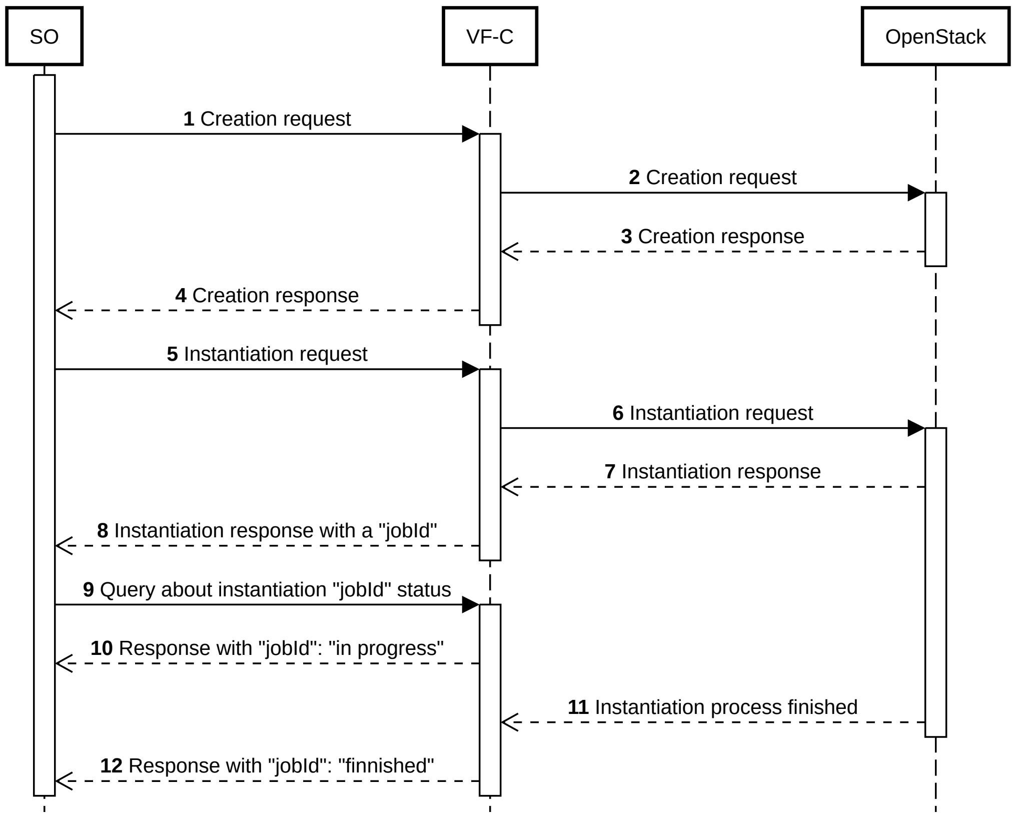

SO
VF-C
OpenStack
1 Creation request
2 Creation request
3 Creation response
4 Creation response
5 Instantiation request
6 Instantiation request
7 Instantiation response
8 Instantiation response with a "jobId"
9 Query about instantiation "jobId" status
10 Response with "jobId": "in progress"
11 Instantiation process finished
12 Response with "jobId": "finnished"

Figure 6.1: Creation and instantiation operations in a successful scenario.

Two main interactions can be identified. On one hand, the communication between SO and VF-C, with the SO ordering lifecycle management operations to the VF-C. On the other hand, the interactions between VF-C and OpenStack, with the VF-C as a controller adapter, playing an important role here bridging these two components.

Of these three participants, the OpenStack component spends more processing time as it is in charge, of the instantiation and termination operations (depending on the scenario of success or error), consisting on creation or removal of virtual links, connection points, launching or finishing the services instances, association or dissociation of floating IP address, etc.

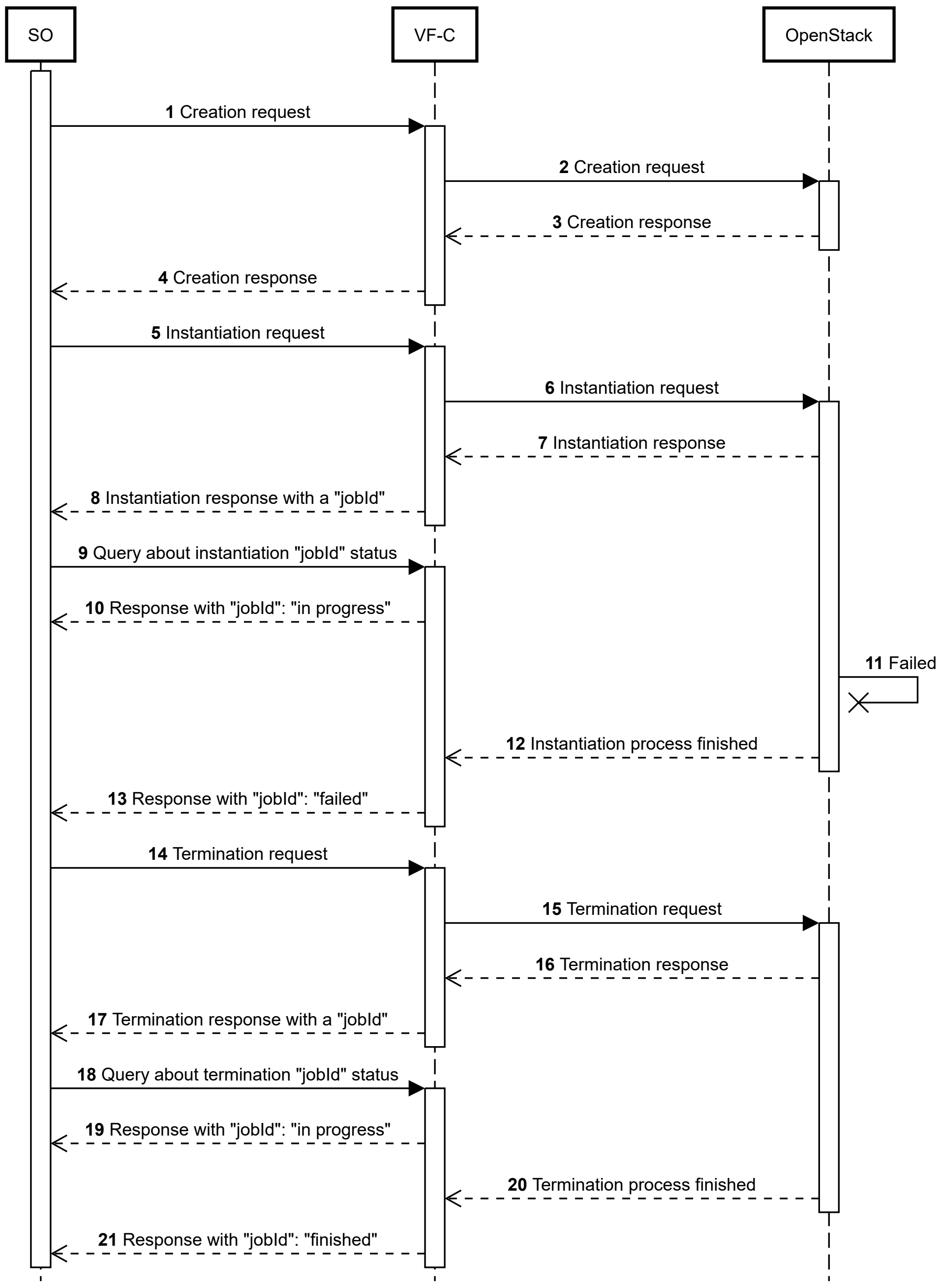

SO
VF-C
OpenStack
1 Creation request
2 Creation request
3 Creation response
4 Creation response
5 Instantiation request
6 Instantiation request
7 Instantiation response
8 Instantiation response with a "jobId"
9 Query about instantiation "jobId" status
10 Response with "jobId": "in progress"
11 Failed
12 Instantiation process finished
13 Response with "jobId": "failed"
14 Termination request
15 Termination request
16 Termination response
17 Termination response with a "jobId"
18 Query about termination "jobId" status
19 Response with "jobId": "in progress"
20 Termination process finished
21 Response with "jobId": "finished"

Figure 6.2: Creation, instantiation and termination operations in an error scenario.

Two scenarios were defined for exprimentation:

- **Success Scenario:** as represented in the sequence diagram of Figure 6.1, comprises the creation (interactions 1 to 4), the instantiation request (interactions 5 to 8) and the respective operation of monitoring the instantiation progress (starts at interaction number 9 and depending on the time that instantiation operation takes, it will perform several queries and responses until finishing at interaction number 12 by acknowledging the “jobId” status of *finnished*);

- **Error Scenario:** as depicted in the sequence diagram of Figure 6.2, comprehends the creation (interactions 1 to 4), the instantiation (interactions 5 to 8) and the operation of monitoring the instantiation progress (same interactions as explained in scenario of success), but in this case, the instantiation of Network Service failed (interaction 11). In this turn, the termination request takes place (interactions 14 to 17) ordering the finish of the NS instance and also tracking the termination progress (between interaction 18 and 19 several queries and responses are performed until the end at interaction number 21 by acknowledging the *finnished* status).

6.1.2 Testbed Environment

In order to achieve a value for these specific operations of the vCDN Edge Node, the tests performed during this analysis were executed in the same testbed environment with two separate test cases, one for ONAP (as previously presented in Section 4.1.3) and another for the vCDN service, both made available in the same physical location.

Table 6.1: Testbed configuration of vCDN Edge Node.

Virtual Machines	**Resources**			
	vCPU	**RAM (GB)**	**HD (GB)**	**OS**
vCDN Edge Node Storage VNF	2	4	20	Ubuntu 18.04
vCDN Edge Node Streamer VNF	2	4	20	Ubuntu 18.04

The testbed environment provided an OpenStack (Queens release) infrastructure containing already deployed instances of the required vCDN service components. In particular, the test case of the vCDN Edge Node component consists of two VNFs, the Storage VNF and the Streamer VNF, both with 2 vCPU and 4 GB of RAM, as described in Table 6.1.

6.2 Determination of Reference Values

This Section presents the used method to obtain results without any Time Delay value defined. For the determination of reference values, a Bash script was developed, which is available in Appendix B, to measure the elapsed execution time and also count the number of requests made to the API of VF-C. The results obtained with two sets of tests, each with 20 samples collected for both considered scenarios, are presented in Table 6.2. This Table allows to analyze the different variables of each operation, such as the average execution time, the standard deviation, the average number of requests and also the overhead size (payload). With this last variable being obtained through the multiplication between the Average Number of Requests and the size of one API request for instantiation (351 bytes) and another for the termination (353 bytes) operation[1].

[1]These values were obtained by using Wireshark open-source packet analyzer application.

Table 6.2: Obtained reference values for both scenarios.

Scenario	**Average Time (s)**	**Standard Deviation (s)**	**Total Samples**	**Average Number of Requests**	**Overhead Size (kB)**
Success					
Monitoring of Instantiation Progress	521.66	76.18	20	906.8	318.27
Error					
Monitoring of Instantiation Progress	131.04	9.95	20	364.1	127.79
Monitoring of Termination Progress	12.78	6.99	20	30.2	10.66

For the success scenario, the script follows the logic of Figure 6.1, however in this case the SO is replaced by the computer terminal that executes the Bash script and the used VF-C interface is an external one to allow this set of tests. In this scenario, for the creation and instantiation operations, the average time was 521.66 s, the standard deviation was 76.18 s, the average number of requests was 906.8 and the total overhead size was 318.27 kB.

In the error test scenario, the developed script follows the logic of Figure 6.2, with the SO also being replaced by the computer terminal that runs the Bash script and the external interface of VF-C was used for this test. The obtained results for the creation and instantiation operations, the average time was 131.04 s and the standard deviation was 9.95 s, with an average number of requests of 364.1 and the total overhead size was 127.79 kB. And for the termination operation, the average time was 12.78 s while the standard deviation was 6.99 s, with the total overhead size of 10.66 kB and the average number of requests was 30.2.

6.3 Obtained Results for Time Delay Definition

This Section presents the results obtained for the tests carried out with different values of Time Delay, defined for both scenarios of success and error. The selected test values were:

- $T_1 = 1\,s$; $T_2 = 2\,s$; $T_3 = 5\,s$; $T_4 = 10\,s$; $T_5 = 15\,s$; $T_6 = 20\,s$; $T_7 = 60\,s$.

Through the results obtained for these Time Delay values, it will be possible to select the value that encompasses the best trade-off, between the number of calls made to the API and the total execution time. The analysis and selection of this value will be further analyzed in Section 6.4. The following subsections present the testing method and the obtained results for both scenarios through the designed workflow.

6.3.1 Testing Method

In the designed workflow, by configuring the Script Task of Time Delay with the values previously selected, and programming the workflow to measure the elapsed execution time and the number of requests made, it was possible to obtain the results here presented.

Each individual test consists of making an API request to the ONAP NBI ordering the instantiation of the service.

Then, through the computer terminal, by accessing to the SO log file, it was possible to follow every step of the workflow and register the results presented in the next subsections.

6.3.2 Success Scenario

The obtained results for this scenario are depicted in the bar chart of Figure 6.3 and summarized in Table 6.3.

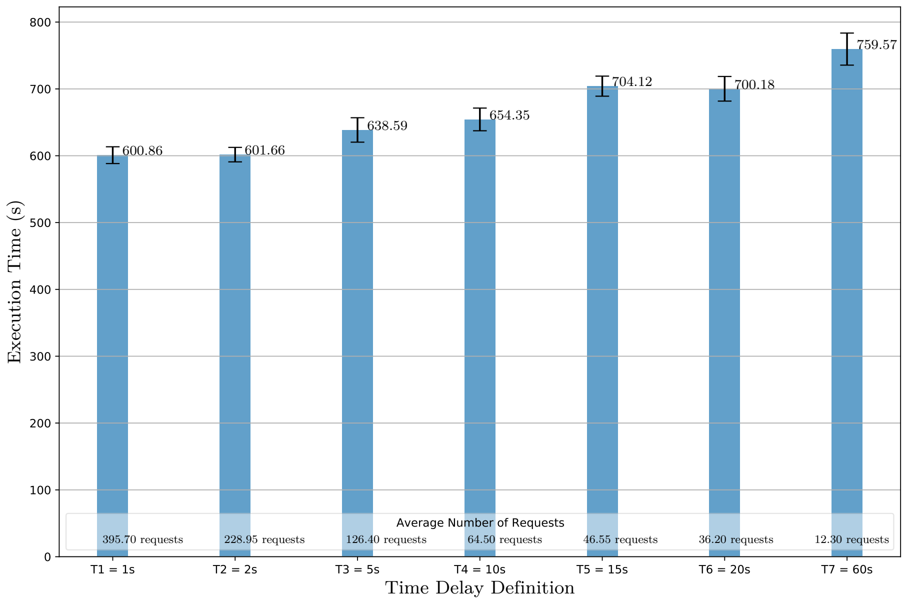


Figure 6.3: Obtained results of monitoring of instantiation progress in a scenario of success.

The lower value of Time Delay, the T_1 value has a mean execution time of 600.86 s and the average number of requests was 396.7. For the higher value of Time Delay, the T_7 value the average execution time was 759.57 s and the average number of requests was 12.3.

Table 6.3: Obtained results of monitoring of instantiation progress in a scenario of success.

Time Delay (s)	**Average Time (s)**	**Standard Deviation (s)**	**Total Samples**	**Average Number of Requests**	**Overhead Size (kB)**
1	600.860	23.050	20	395.70	138.89
2	601.657	23.143	20	228.95	80.36
5	638.589	39.063	20	126.40	44.37
10	654.352	38.259	20	64.50	22.64
15	704.122	32.094	20	46.55	16.34
20	700.175	39.358	20	36.20	12.71
60	759.571	57.202	20	12.30	4.32

6.3.3 Error Scenario

6.3.3.1 Monitoring of Instantiation Progress

The obtained results for the operations of monitoring the instantiation progress in an error scenario are depicted in the bar chart of Figure 6.4 and summarized in Table 6.4.

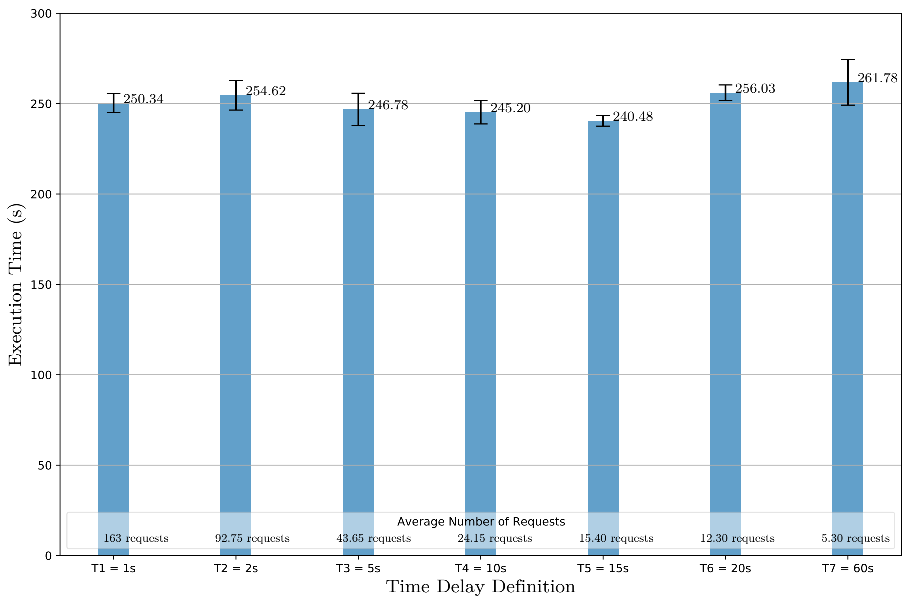


Figure 6.4: Obtained results of monitoring of instantiation progress in an error scenario.

The T_1 value has a mean execution time of 250.34 s and the average number of requests was 163. The T_7 value the average execution time was 261.78 s and the average number of requests was 5.3.

Table 6.4: Obtained results of monitoring of instantiation progress in an error scenario.

Time Delay (s)	**Average Time (s)**	**Standard Deviation (s)**	**Total Samples**	**Average Number of Requests**	**Overhead Size (kB)**
1	250.337	11.289	20	163	57.21
2	254.619	8.675	20	92.75	32.55
5	246.779	12.135	20	43.65	15.32
10	245.203	13.732	20	24.15	8.48
15	240.475	6.257	20	15.40	5.41
20	256.028	9.243	20	12.30	4.32
60	261.781	26.960	20	5.30	1.86

6.3.3.2 Monitoring of Termination Progress

The obtained results for the operations of monitor the termination progress in an error scenario are depicted in the bar chart of Figure 6.5 and summarized in Table 6.5.

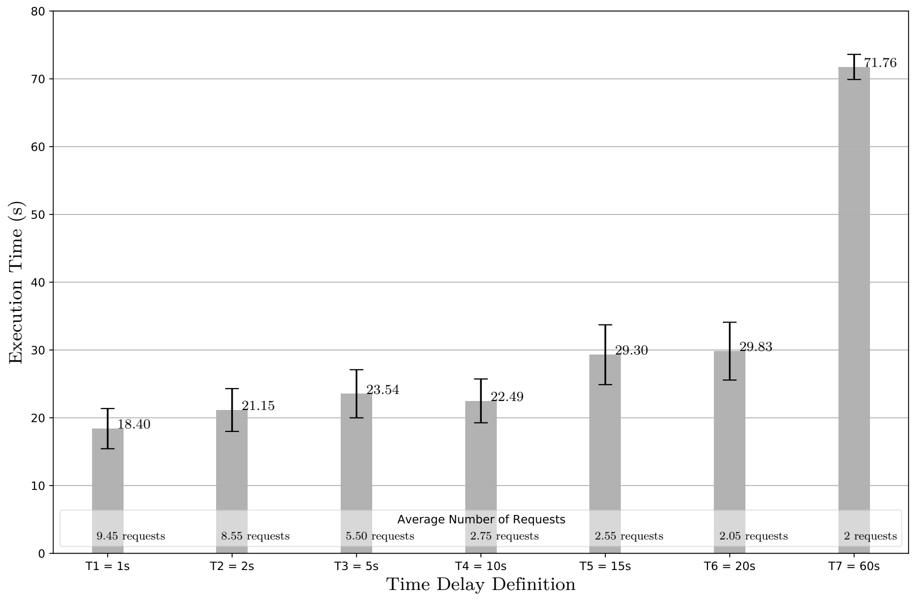


Figure 6.5: Obtained results of monitoring of termination progress in an error scenario.

The lower value of Time Delay, the T_1 value has a mean execution time of 18.40 s and the average number of requests was 9.45. For the higher value of Time Delay, the T_7 value the average execution time was 71.76 s and the average number of requests was 2.

Table 6.5: Obtained results of monitoring of termination progress in an error scenario.

Time Delay (s)	**Average Time (s)**	**Standard Deviation (s)**	**Total Samples**	**Average Number of Requests**	**Overhead Size (kB)**
1	18.400	6.334	20	9.45	3.34
2	21.146	6.201	20	8.55	3.02
5	23.542	7.581	20	5.50	1.94
10	22.493	6.605	20	2.75	0.97
15	29.304	9.428	20	2.55	0.90
20	29.828	9.109	20	2.05	0.72
60	71.755	3.955	20	2	0.71

6.4 Results Analysis

This Section provides the analysis of the obtained results previously presented in Section 6.2 and 6.3 for the selection of the Time Delay value.

6.4.1 Observations

As observed in both sequence diagram of Figure 6.1 and 6.2, a great part of processing time is spent by OpenStack during the operations to initialize and configure the required VNF instances. And also in an opposite way, during the shut down of VNF instances when termination request takes place. The remaining operating time is used in message exchanges between the SO and VF-C, and between the VF-C and OpenStack. The Activities performed by the workflow also spend time on execution and processing but are not significant.

As mentioned at the beginning of this Chapter, the main goal is to improve the quality of the designed workflow by evaluating an value with the best trade-off for the configuration of the Time Delay Script Task operations.

Through the analysis of the Tables 6.3, 6.4 and 6.5, it is possible to make the following high-level observations: as the value set for the Time Delay increases, the average number of requests realized decreases. As well as the size of the associated overhead also decreases. Besides, as the value set for the Time Delay increases, the workflow completion time also increases, as would be expected, since the waiting time is increasing.

6.4.2 Time Delay Selection

The method used for the Time Delay selection focuses on the analysis of only two values of Time Delay: T_2=2 s and T_3=5 s, thus avoiding T_0 and T_1 as they have a high average number of orders placed and have lower waiting values. And also avoiding the times of T_4, T_5, T_6 and T_7 as they have Time Delays values that greatly increase the workflow execution time.

Therefore, considering only the values of execution time and the number of average requests, in a successful scenario for the monitor of instantiation progress operation, T_2 value has an average execution time of 601.66 s and an average number of requests of 228.95. Meanwhile, T_3 value has an average execution time of 638.59 s and an average number of requests of 126.40.

For the operation of monitoring of instantiation progress, in an error scenario, T_2 value has an average execution time of 254.62 s and the average number of requests is 92.75. In contrast, T_3 value has an average execution time of 245.29 s and an average number of requests of 43.65. Finally, for the operation of monitoring of termination progress, T_2 value has an average execution time of 21.15 s and an average number of requests of 8.55. In turn, T_3 value has an average execution time of 23.54 s and an average number of requests of 5.50.

After this analysis and taking into account the considered reasons for its selection, the T_3 value was used. For presenting the lowest average number of requests made and being also the most balanced among the other selected values.

As a final observation, the definition of Time Delay value brings value to this work by increasing the quality of the designed workflow and also proved its functional validation. This study was not defined as a goal of this work but helped to complement the quality of these operations. However, the high standard deviation has shown that the conditions of the system vary significantly. Furthermore, the values here obtained are only valid for the used testbed environment.

Chapter 7

Final Remarks

This Chapter comprehends the entire work of this Dissertation, by describing the work performed and its value, as well as possible future improvements that can be made.

7.1 Conclusions

The main objective of this Dissertation was the integration of services developed for 5G networks, by enabling the essential lifecycle management operations using a reference open-source orchestration platform.

In order to address the objective of this work, various phases were identified. The first phase comprised the study and the understanding of several background concepts, more specifically about the Camunda BPM platform. From all components of Camunda BPM, the Camunda Modeler had a special relevance in this work, enabling the design and implementation of the orchestration logic of the workflow to achieve process automation.

The second phase included the comprehensive study of the ONAP by identifying the necessary components and their operations. This phase also included the validation of the considered orchestration platform, by integrating a vCDN service in the scope of a closed-loop scenario in order to solve a network problem through the scale-out operation of the vCDN service. The SO and VF-C components were highly relevant in this phase. On one hand, the SO component performed the service orchestration logic by taking advantage of the Camunda Process Engine. On the other hand, the VF-C enabled the integration between the vCDN service and the orchestration platform, since they are in line with ETSI NFV-MANO reference architecture.

After the successful deployment and validation of the vCDN service lifecycle management, the attention turned to the improvement of the quality of the designed workflow arises. The results of the tests carried out made it possible to evaluate a better value for these operations, preventing network overloads with excessive requests, and also avoiding time wastes during the operation execution.

In conclusion, this Dissertation showed that ONAP is an excellent open-source NSO solution, and its ETSI-alignment increases the interoperability with different vendors and reduces the time-to-market of the new products. Besides, the design and development of the scale-out operation were successfully validated enabling the orchestration of the vCDN service, which directly contributed to the 5G Mobilizer project.

7.2 Future Work

In terms of future work, some considerations are described as follows:

- **Error Mapping:** implementation of an error map mechanism, which intends to inform the developer of the location of any possible error that may occur during the execution of the workflow;

- **Handling of repeated requests:** implementation of the ability to evaluate repeated requests in the designed workflow. Assuming a system error scenario, in which for some reason the component responsible for the request validation is not working well, and for example, it accepts two repeated requests. In this context, assigning this functionality to the workflow will allow automated management for this type of situation, in the process of validation and decision make of only one request for the orchestration of the desired service. As a result, the orchestrator will be more independent and resilient.

Appendix A

ONAP ETSI NFV API

ETSI NFV-SOL 005 defined RESTful API for Os-Ma-nfvo reference point, shared between OSS and NFVO functions. Mapping this architecture to ONAP components. The SO represents the OSS functionality as it oversees end-to-end services and VF-C represents the NFVO functionality as it performs the orchestration of network services. By aligning the interface between SO and VF-C to ETSI NFV-SOL 005, any external NFVO can use ONAP SO for the service orchestration layer.

This Appendix provides the RESTful API specification and the associated request payload for the NSD Management interface, VNF Package Management interface and Network Service Lifecycle Management interface operations. More details about SO offered and consumed APIs can be found in [126, 137].

A.1 Network Service Descriptor Management Interface

The Network Service Descriptor Management interface allows to subscribe, query subscription information, terminate, and notify. The API request that was used in the interaction between the SO and NFVO component is presented in the following subsection.

A.1.1 Get Network Service Descriptor Resources

Table A.1: API Request Definition of Get information about NSD resources.

Interface Definition	**Description**
Operation Type	GET
Content-Type	application/json
Headers	Basic Authentication
URI	http://{hostname}:30280/api/nsd/v1/ns_descriptors

Table A.2: API Response Payload of Get NSD resources.

Attribute	**Data Type**	**Description**
id	String	Uniquely identifies this instance of Network Service
nsdId	String	ID of Network Service Descriptor
nsdName	String	Name of the Network Service Descriptor
nsdInvariantId	String	ID of Network Service Descriptor
vnfPkgIds	String	Identifies the VNF Packages IDs for the VNFD

A.2 VNF Package Management Interface

The VNF Package Management interface allows to subscribe, fetch VNF Package artifacts, query subscription information and terminate. The API request that was used in the interaction between SO and NFVO component is presented in the following subsection.

A.2.1 Get Individual VNF Package Information

Table A.3: API Request Definition of Get information about an individual VNF package.

Interface Definition	**Description**
Operation Type	GET
Content-Type	application/json
Headers	Basic Authentication
URI	http://{hostname}:30280/api/vnfpkgm/v1/vnf_packages/{vnfPkgId}

Table A.4: API Response Payload of Get information about an individual VNF package.

Attribute	**Data Type**	**Description**
id	String	Uniquely identifies the VNF Package
vnfdId	String	ID of the VNF Descriptor Package resource

A.3 Network Service Lifecycle Management Interface

The Network Service Lifecycle Management interface allows various operations, such as create, delete, get, instantiate and terminate Network Services. These API requests were used in the interaction between SO and NFVO component, and are presented in the following subsections.

A.3.1 Create Network Service

Table A.5: API Request Definition of Create NS.

Interface Definition	**Description**
Operation Type	POST
Content-Type	application/json
Headers	Basic Authentication
URI	http://{hostname}:30280/api/nslcm/v1/ns/

Table A.6: API Request Payload of Create NS.

Attribute	**Data Type**	**Description**
context	Object	Information on the Context
globalCustomerId	String	Global Customer ID used to uniquely identify customer
serviceType	String	Value defined by orchestration to identify the service
nsName	String	Name of the Network Service
csarId	String	Information regarding workflow control parameters
description	String	Description

Table A.7: API Response Payload of Create NS.

Attribute	**Data Type**	**Description**
nsInstanceId	String	Uniquely identifies this instance of a service

A.3.2 Instantiate Network Service

Table A.8: API Request Definition of Instantiate NS.

Interface Definition	**Description**
Operation Type	POST
Content-Type	application/json
Headers	Basic Authentication
URI	http://{hostname}:30280/api/nslcm/v1/ns/{nsInstanceId}/instantiate

Table A.9: API Request Payload of Instantiate NS.

Attribute	**Data Type**	**Description**
nsInstanceId	String	Uniquely identifies this instance of Network Service
additionalParamForNs	String	additionalParamForNs
LocationConstraints	Array	LocationConstraints (vnfProfileId and vimId)

Table A.10: API Response Payload of Instantiate NS.

Attribute	**Data Type**	**Description**
jobId	String	Job ID

A.3.3 Monitoring Instantiation/Termination Progress

Table A.11: API Request Definition of Get operation progress.

Interface Definition	**Description**
Operation Type	GET
Content-Type	application/json
Headers	Basic Authentication
URI	http://{hostname}:30280/api/nslcm/v1/jobs/{jobId}

Table A.12: API Response Payload of Get NS.

Attribute	**Data Type**	**Description**
nsInstanceId	String	Uniquely identifies this instance of Network Service
nsName	String	Name of the Network Service
description	String	Description
nsdId	String	ID of Network Service
nsState	String	State of Network Service

A.3.4 Get Network Service

Table A.13: API Request Definition of Get NS.

Interface Definition	**Description**
Operation Type	GET
Content-Type	application/json
Headers	Basic Authentication
URI	http://{hostname}:30280/api/nslcm/v1/ns/

Table A.14: API Response Payload of Get NS.

Attribute	**Data Type**	**Description**
nsInstanceId	String	Uniquely identifies this instance of Network Service
nsName	String	Name of the Network Service
description	String	Description
nsdId	String	ID of Network Service
nsState	String	State of Network Service

A.3.5 Terminate Network Service

Table A.15: API Request Definition of Terminate NS.

Interface Definition	**Description**
Operation Type	POST
Content-Type	application/json
Headers	Basic Authentication
URI	http://{hostname}:30280/api/nslcm/v1/ns/{nsInstanceId}/terminate

Table A.16: API Request Payload of Terminate NS.

Attribute	**Data Type**	**Description**
nsInstanceId	String	Uniquely identifies this instance of Network Service
gracefulTerminationTimeout	String	gracefulTerminationTimeout
terminationType	String	terminationType

Table A.17: API Response Payload of Terminate NS.

Attribute	**Data Type**	**Description**
jobId	String	Job ID

Appendix B

Service Response Time

This Appendix includes the Bash script developed to obtain the duration of the elapsed time and also the maximum number of requests made between the SO and the VF-C.

B.1 Bash Script

```bash
#!/bin/bash
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
# Title:       Script to measure the Service Response Time between SO and VF-C
# Description: This script measures time and number of API requests
# Operations:  Create, Instantiate and Terminate NS
# Author:      Tiago Amaral
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
# Installation of Curl and jq (external library for Json parser):
# sudo apt update
# sudo apt install -y curl
# sudo apt-get install jq

for i in {1..20}
do

# Set Counter Request to 0
countRequestInstantiate=0
# Start Time Counter:
res1=$(date +%s.%N)

# # # # # # # # # # # # # # # # Create NS # # # # # # # # # # # # # # # # # # # # #
printf "\n------ Begin: Create Network Service -----------"
curl -s --location --request POST 'http://192.168.1.52:30280/api/nslcm/v1/ns' \
-w "\n\tResponse Code is: %{http_code}\n" \
-o responseGetCreateNS.json \
--header 'accept: application/json' \
--header 'cache-control: no-cache' \
--header 'content-type: application/json' \
--data-raw '{
    "context": {
        "globalCustomerId": "Altran",
        "serviceType": "SERVICE_ATRAN_DEMO"
    },
    "csarId": "0d58bde6-5aec-4bda-af8d-7d6c85f2db59",
    "description": "description",
    "nsName": "NS_NODE_STREAMER1"
}'
# Obtain nsIntanceId:
nsInstanceId=$(jq -r '.nsInstanceId' responseGetCreateNS.json)
printf "\n\tNS Instance ID is: $nsInstanceId"
# Increment Counter:
countRequestInstantiate=$(($countRequestInstantiate+1))
printf "\n------ Finish: Create Network Service -----------\n"

# # # # # # # # # # # # # #  Instantiate NS # # # # # # # # # # # # # # # # # # # #
printf "\n------ Begin: Instantiate Network Service --------"
curl -s --location --request POST "http://192.168.1.52:30280/api/nslcm/v1/ns/
    $nsInstanceId/instantiate" \
-w "\n\tResponse Code is: %{http_code}\n" \
-o responseInstantiate.json \
--header 'accept: application/json' \
--header 'cache-control: no-cache' \
--header 'Content-Type: application/json' \
--data-raw '{
    "additionalParamForNs": {
        "sdnControllerId": "2",
        "vcdn_node_storage_id": "687980890890",
        "vcdn_engine_address": "192.168.1.74"
    },
    "locationConstraints": [
        {
            "vnfProfileId": "ab40934c-8b92-4514-ad6e-e84f9f4edc47",
            "locationConstraints": {
                "vimId": "ONAP_RegionOne"
            }
        },
        {
            "vnfProfileId": "04db9544-5e13-4638-9ca0-1ce8b1eab15e",
            "locationConstraints": {
                "vimId": "ONAP_RegionOne"
            }
        }
    ]
}'
# Obtain jobId from NS Intance:
jobId=$(jq -r '.jobId' responseInstantiate.json)
printf "\n\n\tJobId is: $jobId"

```

```bash
# Increment Counter from last request and for the next one:
countRequestInstantiate=$(($countRequestInstantiate+2))

printf "\n\n------ Check Instantiate Progress --------\n"
while true
do
jobId=$(jq -r '.jobId' responseInstantiate.json)
printf "\n\n\tJobId is: $jobId"
url='http://192.168.1.52:30280/api/nslcm/v1/jobs/'
urlFinal="$url$jobId"
printf "\n\n\tURL for Check Instantiate Progress is:\n\n\t$urlFinal"
  curl -s --location --request GET "$urlFinal" \
  -w "\n\n\tResponse Code is: %{http_code}\n" \
  -o responseInstProgress.json \
  --header 'accept: application/json' \
  --header 'content-type: application/json' \
  --header 'cache-control: no-cache' \
  --header 'connection: keep-alive' \
  --data-raw ''
  # Obtain operationStatus:
  operationStatus=$(jq -r '.responseDescriptor.status' responseInstProgress.json)
  # Set condition for break:
  if [ "$operationStatus" != "processing" ]
  then
    printf "\n\n\tThe operationStatus is: $operationStatus" >> instantiate.txt
    break
  # Set condition for continue:
  else
    # Increment Counter:
    countRequestInstantiate=$(($countRequestInstantiate+1))
    printf "\n\n\tThe operationStatus is: processing..."
    continue
  fi
done
printf "\n\n------ Finish: Check Instantiate Progress --------\n"
printf "\n\n------ Finish: Instantiate Network Service --------\n"
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
res2=$(date +%s.%N)
dt=$(echo "$res2 - $res1" | bc)
dd=$(echo "$dt/86400" | bc)
dt2=$(echo "$dt-86400*$dd" | bc)
dh=$(echo "$dt2/3600" | bc)
dt3=$(echo "$dt2-3600*$dh" | bc)
dm=$(echo "$dt3/60" | bc)
ds=$(echo "$dt3-60*$dm" | bc)

LC_NUMERIC=C printf "\n\tTotal runtime (instantiate): %d:%02d:%02d:%02.3f\n" $dd
    $dh $dm $ds >> instantiate.txt
printf "\n\tCounter Request value is: $countRequestInstantiate" >> instantiate.txt
```

```
# # # # # # # # # # # # # #  Terminate NS # # # # # # # # # # # # # # # # # # # # # #
res3=$(date +%s.%N)
# Set Counter Request Terminate to 0:
countRequestTerminate=0

printf "\n------ Begin: Terminate Network Service -----------"
# Obtain nsInstanceId variable to terminate:
nsInstanceId=$(jq -r '.nsInstanceId' responseGetCreateNS.json)
printf "\n\n\tNS Instance ID is: $nsInstanceId"
url='http://192.168.1.52:30280/api/nslcm/v1/ns/'
action='/terminate'
urlTerminate="$url$nsInstanceId$action"
printf "\n\n\tURL for Termination Request is:\n\n\t$urlTerminate"
curl -s --location --request POST "$urlTerminate" -w "\n\tResponse Code is: %{
    http_code}\n" -o responseTerminate.json \
--header 'accept: application/json' \
--header 'cache-control: no-cache' \
--header 'content-type: application/json' \
--data-raw '{
    "gracefulTerminationTimeout": 600,
    "terminationType": "FORCEFUL"
}'
# Obtain jobId from NS Intance:
jobIdTerminate=$(jq -r '.jobId' responseTerminate.json)
printf "\n\n\tJobId is: $jobIdTerminate"
# Increment countRequestTerminate from last request and for the next one:
countRequestTerminate=$(($countRequestTerminate+2))
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
printf "\n\n------ Check Terminate Progress --------\n"
while true
do
jobIdTerminate=$(jq -r '.jobId' responseTerminate.json)
printf "\n\n\tJobId is: $jobIdTerminate"

url='http://192.168.1.52:30280/api/nslcm/v1/jobs/'
urlTermProgress="$url$jobIdTerminate"
printf "\n\n\tURL for Check Terminate Progress is:\n\n\t$urlTermProgress"

  curl -s --location --request GET "$urlTermProgress" -w "\n\tResponse Code is: %{
    http_code}\n" -o responseTermProgress.json \
  --header 'accept: application/json' \
  --header 'content-type: application/json' \
  --header 'cache-control: no-cache' \
  --header 'connection: keep-alive' \
  --data-raw ''

  # Obtain operationStatus:
  operationStatus=$(jq -r '.responseDescriptor.status' responseTermProgress.json)

```

```bash
  # Set condition for break:
  if [ "$operationStatus" != "processing" ]
  then
    printf "\n\n\tThe operationStatus is: $operationStatus" >> terminate.txt
    break
    # Set condition for continue:
  else
    # Increment Counter:
    countRequestTerminate=$(($countRequestTerminate+1))
    printf "\n\n\tThe operationStatus is: processing..."
    continue
  fi
done
printf "\n\n------ Finish: Check Terminate Progress --------\n"
printf "\n\n------ Finish: Terminate Network Service --------\n"

res4=$(date +%s.%N)
dt=$(echo "$res4 - $res3" | bc)
dd=$(echo "$dt/86400" | bc)
dh=$(echo "$dt2/3600" | bc)
dt3=$(echo "$dt2-3600*$dh" | bc)
dm=$(echo "$dt3/60" | bc)
ds=$(echo "$dt3-60*$dm" | bc)
LC_NUMERIC=C printf "\n\tTotal runtime (terminate): %d:%02d:%02d:%02.3f\n" $dd $dh
    $dm $ds >> terminate.txt
printf "\n\tCounter Request FINAL value for termination operation is:
    $countRequestTerminate" >> terminate.txt
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
done
# EOF
```

Listing B.1: Script for measuring elapsed time and number of requests

References

[1] K. Kusume and M. Fallgren, "Updated scenarios, requirements and KPIs for 5G mobile and wireless system with recommendations for future investigations (D1.5)," METIS, Tech. Rep., May. 2015, last accessed on 12/04/2020. [Online]. Available: www.metis2020.com/wp-content/uploads/deliverables/METIS_D1.5_v1.pdf

[2] 5G-PPP Architecture Working Group, "View on 5G Architecture v. 3.0," 5G PPP Initiative, Tech. Rep., Jun. 2019. [Online]. Available: www.5g-ppp.eu/wp-content/uploads/2019/07/5G-PPP-5G-Architecture-White-Paper_v3.0_PublicConsultation.pdf

[3] Ericsson, "White Paper: 5G Systems – Enabling the transformation of industry and society," Ericsson, Tech. Rep., Jan. 2017, last accessed on 12/04/2020. [Online]. Available: www.ericsson.com/49daeb/assets/local/reports-papers/white-papers/wp-5g-systems.pdf

[4] 5GO, "Project 5GO," last accessed on 19/04/2020. [Online]. Available: https://5go.pt/en/

[5] E. Dahlman, S. Parkvall, and J. Sköld, *4G: LTE/LTE-Advanced for Mobile Broadband*, 2nd ed. Academic Press, 2014.

[6] A. A. Barakabitze, A. Ahmad, R. Mijumbi, and A. Hines, "5G network slicing using SDN and NFV: A survey of taxonomy, architectures and future challenges," *Computer Networks*, vol. 167, p. 40, 2020. [Online]. Available: http://doi.org/dv29

[7] ITU-R, "Recommendation ITU-R M.2370-0 -IMT traffic estimates for the years 2020 to 2030," M Series, Tech. Rep., 2015. [Online]. Available: www.itu.int/ITU-R/go/patents/en

[8] E. Dahlman, S. Parkvall, and J. Sköld, *5G NR: The Next Generation Wireless Access Technology*. Mara Conner, 2018.

[9] ITU-R, "ITU-R FAQ on IMT," 2019, last accessed on 20/04/2020. [Online]. Available: www.itu.int/en/ITU-R/Documents/ITU-R-FAQ-IMT.pdf

[10] ITU, "ITU global standard for international mobile telecommunications," 2020, last accessed on 20/04/2020. [Online]. Available: www.itu.int/en/ITU-R/study-groups/rsg5/rwp5d/imt-adv/Pages/default.aspx

[11] ITU-R, "Recommendation ITU-R M.2083-0 - IMT Vision – Framework and overall objectives of the future development of IMT for 2020 and beyond," M Series, Tech. Rep., 2015. [Online]. Available: www.itu.int/ITU-R/go/patents/en

[12] E. Guttman, “5G Standardization in 3GPP,” 2018, last accessed on 20/04/2020. [Online]. Available: www.itu.int/en/ITU-T/Workshops-and-Seminars/201807/Documents/3_Erik_Guttman.pdf

[13] ETSI, “ETSI Worldwide - Global Business Standards Collaboration,” 2020, last accessed on 20/04/2020. [Online]. Available: www.etsi.org/about/etsi-worldwide

[14] L. Casaccia, “Understanding 3GPP – starting with the basics,” 2017, last accessed on 21/04/2020. [Online]. Available: www.qualcomm.com/news/onq/2017/08/02/understanding-3gpp-starting-basics

[15] 3GPP, “Releases,” 2020, last accessed on 21/04/2020. [Online]. Available: www.3gpp.org/specifications/releases

[16] 3GPP, “Feasibility Study on New Services and Markets Technology Enablers,” 3rd Generation Partnership Project, Tech. Rep. 22.891 - Release 14.1.0, Stage 1, Jun. 2016. [Online]. Available: www.3gpp.org/DynaReport/22891.htm

[17] 3GPP, “Feasibility Study on New Services and Markets Technology Enablers for Massive Internet of Things,” 3rd Generation Partnership Project, Tech. Rep. 22.861 - Release 14.1.0, Stage 1, Sep. 2016. [Online]. Available: www.3gpp.org/DynaReport/22861.htm

[18] 3GPP, “Feasibility Study on New Services and Markets Technology Enablers for Critical Communications,” 3rd Generation Partnership Project, Tech. Rep. 22.862 - Release 14.1.0, Stage 1, Oct. 2016. [Online]. Available: www.3gpp.org/DynaReport/22862.htm

[19] 3GPP, “Feasibility Study on New Services and Markets Technology Enablers - enhanced Mobile Broadband,” 3rd Generation Partnership Project, Tech. Rep. 22.863 - Release 14.1.0, Stage 1, Sep. 2016. [Online]. Available: www.3gpp.org/DynaReport/22863.htm

[20] 3GPP, “Feasibility Study on New Services and Markets Technology Enablers - Network Operation,” 3rd Generation Partnership Project, Tech. Rep. 22.864 - Release 14.1.0, Stage 1, Sep. 2016. [Online]. Available: www.3gpp.org/DynaReport/22864.htm

[21] 3GPP, “Service requirements for V2X services,” 3rd Generation Partnership Project, Tech. Rep. 22.185 - Release 14.1.0, Stage 1, Jun. 2016. [Online]. Available: www.3gpp.org/DynaReport/22185.htm

[22] 3GPP, “Summary of Rel-15 Work Items,” 3rd Generation Partnership Project, Tech. Rep. 21.915 - Release 15.0, Oct. 2019. [Online]. Available: www.3gpp.org/DynaReport/21915.htm

[23] ETSI, “Mobile Technologies - 5G,” 2020, last accessed on 20/04/2020. [Online]. Available: www.etsi.org/technologies/5g

[24] 3GPP, “System architecture for the 5G System,” 3rd Generation Partnership Project, Tech. Rep. 23.501 - Release 16.4.0, Stage 2, Mar. 2020. [Online]. Available: www.3gpp.org/DynaReport/23501.htm

[25] 3GPP, "Study on new radio access technology: Radio access architecture and interfaces," 3rd Generation Partnership Project, Tech. Rep. 38.801 - Release 14.0, Apr. 2017. [Online]. Available: www.3gpp.org/DynaReport/38801.htm

[26] 5G Americas, "5G Network Transformation - White Paper," 5G Americas, Tech. Rep., Jul. 2017. [Online]. Available: www.5gamericas.org/wp-content/uploads/2019/07/5G_Network_Transformation_Final.pdf

[27] 5G Americas, "5G and the Cloud - White Paper," 5G Americas, Tech. Rep., Dec. 2019. [Online]. Available: www.5gamericas.org/wp-content/uploads/2019/12/5G-Americas_5G-and-the-Cloud..pdf

[28] 3GPP, "5G System; Technical Realization of Service Based Architecture," 3rd Generation Partnership Project, Tech. Rep. 29.500 - Release 16.3.0, Stage 3, Mar. 2019. [Online]. Available: www.3gpp.org/DynaReport/29500.htm

[29] R. El Hattachi and J. Erfanian, "5G White Paper by NGMN Alliance v.1.0," 5G Initiative, Tech. Rep., Feb. 2015. [Online]. Available: www.ngmn.org/wp-content/uploads/NGMN_5G_White_Paper_V1_0.pdf

[30] I. Afolabi, T. Taleb, K. Samdanis, A. Ksentini, and H. Flinck, "Network slicing and softwarization: A survey on principles, enabling technologies, and solutions," *IEEE Communications Surveys and Tutorials*, vol. 20, no. 3, pp. 2429–2453, 2018. [Online]. Available: http://doi.org/gfjx6d

[31] J. Ordonez-Lucena, P. Ameigeiras, D. Lopez, J. J. Ramos-Munoz, J. Lorca, and J. Folgueira, "Network Slicing for 5G with SDN/NFV: Concepts, Architectures, and Challenges," *IEEE Communications Magazine*, vol. 55, no. 5, pp. 80–87, May. 2017. [Online]. Available: http://doi.org/dv2x

[32] F. Zarrar Yousaf, M. Bredel, S. Schaller, and F. Schneider, "NFV and SDN — Key Technology Enablers for 5G Networks," *IEEE Journal*, vol. 35, no. 11, pp. 2468–2478, Nov. 2017. [Online]. Available: http://doi.org/dv2z

[33] 3GPP, "NR; NR and NG-RAN Overall Description," 3rd Generation Partnership Project, Tech. Rep. 38.300 - Release 16.1.0 Stage 2, Apr. 2020. [Online]. Available: www.3gpp.org/DynaReport/38300.htm

[34] M. A. Habibi, M. Nasimi, B. Han, and H. D. Schotten, "A Comprehensive Survey of RAN Architectures Toward 5G Mobile Communication System," *IEEE Access*, vol. 7, pp. 70 371–70 421, 2019. [Online]. Available: http://doi.org/dwh7

[35] 3GPP, "NR; Physical layer; General description," 3rd Generation Partnership Project, Tech. Rep. 38.201 - Release 16.0, Dec. 2019. [Online]. Available: www.3gpp.org/DynaReport/38201.htm

[36] 3GPP, "NR; Physical channels and modulation," 3rd Generation Partnership Project, Tech. Rep. 38.211 - Release 16.1, Mar. 2019. [Online]. Available: www.3gpp.org/DynaReport/38211.htm

[37] 3GPP, "NR; Multiplexing and channel coding," 3rd Generation Partnership Project, Tech. Rep. 38.212 - Release 16.1, Mar. 2019. [Online]. Available: www.3gpp.org/DynaReport/38212.htm

[38] M. Dohler and T. Nakamura, *5G Mobile and Wireless Communications Technology*, 1st ed., A. Osseiran, J. F. Monserrat, and P. Marsch, Eds. Cambridge University Press, 2016.

[39] ETSI, "Network Functions Virtualisation," 2020, last accessed on 28/04/2020. [Online]. Available: www.etsi.org/technologies/nfv

[40] W. Stallings, F. Agboma, and S. Jelassi, *Foundations of Modern Networking: SDN, NFV, QoE, IoT, and Cloud*, ser. The William Stallings books on computer and data communications technology. Pearson, 2015.

[41] M. Chiosi and et al., "ETSI White Paper: NFV - An Introduction, Benefits, Enablers, Challenges & Call for Action," European Telecommunications Standards Institute, Tech. Rep., Oct. 2012, presented at the "SDN and OpenFlow World Congress", Darmstadt - Germany. [Online]. Available: http://portal.etsi.org/NFV/NFV_White_Paper.pdf

[42] ETSI, "GS NFV 002 - V1.2.1 - Network Functions Virtualisation (NFV); Architectural Framework," European Telecommunications Standards Institute, Tech. Rep., Dec. 2014. [Online]. Available: www.etsi.org/standards

[43] ONF, "About the ONF| Mission, Members, Training, Partners," last accessed on 27/04/2020. [Online]. Available: www.opennetworking.org/mission/

[44] ONF, "TR-502: SDN architecture," Open Networking Foundation, Tech. Rep. Issue 1, Jun. 2014. [Online]. Available: www.opennetworking.org/software-defined-standards/archives

[45] ONF, "TR-521: SDN Architecture," Open Networking Foundation, Tech. Rep. Issue 1.1, Oct. 2016. [Online]. Available: www.opennetworking.org/software-defined-standards/archives

[46] ETSI, "GS NFV-EVE 005 - V1.1.1 - Network Functions Virtualisation (NFV); Ecosystem; Report on SDN Usage in NFV Architectural Framework," European Telecommunications Standards Institute, Tech. Rep., Dec. 2015. [Online]. Available: www.etsi.org/standards

[47] ONF, "TR-518: Relationship of SDN and NFV," Open Networking Foundation, Tech. Rep. Issue 1, Oct. 2015. [Online]. Available: www.opennetworking.org/software-defined-standards/archives

[48] C. Chappell, "White Paper - The Evolution to Cloud-Native NFV: Early adoption brings benefits with a flexible approach," Analysys Mason, Tech. Rep., Nov. 2017. [Online]. Available: www.analysysmason.com

[49] 5G-PPP, "White Paper - From Webscale to Telco, the Cloud Native Journey," 5G Infrastructure Public Private Partnership, Tech. Rep., Jul. 2018. [Online]. Available: https://5g-ppp.eu

[50] 5G-PPP, "White Paper - Cloud-Native and Verticals' services," 5G Infrastructure Public Private Partnership, Tech. Rep., Aug. 2019. [Online]. Available: https://5g-ppp.eu

[51] ETSI, "GS MEC 002 - V2.1.1 - Multi-access Edge Computing (MEC); Use Cases and Requirements," European Telecommunications Standards Institute, Tech. Rep., Oct. 2018, phase 2. [Online]. Available: www.etsi.org/standards

[52] G. A. Carella, M. Pauls, T. Magedanz, M. Cilloni, P. Bellavista, and L. Foschini, "Prototyping NFV-based Multi-access Edge Computing in 5G ready Networks with Open Baton," in *IEEE Conference on Network Softwarization (NetSoft)*, Jul. 2017. [Online]. Available: http://doi.org/dv23

[53] S. Kekki and et al., "ETSI White Paper: MEC in 5G networks," European Telecommunications Standards Institute, Tech. Rep. 28, Jun. 2018. [Online]. Available: www.etsi.org

[54] A. Reznik and et al., "MEC in an Enterprise Setting: A Solution Outline," European Telecommunications Standards Institute, Tech. Rep. 30, Sep. 2018. [Online]. Available: www.etsi.org

[55] ETSI, "GS MEC 003 - V2.1.1 - Multi-access Edge Computing (MEC); Framework and Reference Architecture," European Telecommunications Standards Institute, Tech. Rep., Jan. 2019. [Online]. Available: www.etsi.org/standards

[56] TeleManagement Forum (TM Forum), "Frameworx 19.5," 2020, last accessed on 05/05/2020. [Online]. Available: http://casewise.tmforum.org/evolve/statics/frameworx/index.html

[57] Altice Labs, S.A., Altran Portugal, S.A., IT, IT Center, Nokia, Onesource, PDM&FC, Ubiwhere, Universidade de Coimbra, "Use cases and requirements for solutions targetting 5G network core," 5GO consortium, Tech. Rep., Mar. 2018, D2.1 - V1.0. [Online]. Available: https://5go.pt/en/results

[58] R. Guerzoni, I. Vaishnavi, D. Perez Caparros, A. Galis, F. Tusa, P. Monti, A. Sganbelluri, G. Biczók, B. Sonkoly, L. Toka, A. Ramos, J. Melián, O. Dugeon, F. Cugini, B. Martini, P. Iovanna, G. Giuliani, R. Figueiredo, L. M. Contreras-Murillo, C. J. Bernardos, C. Santana, and R. Szabo, "Analysis of end-to-end multi-domain management and orchestration frameworks for software defined infrastructures: an architectural survey," *Transactions on Emerging Telecommunications Technologies*, vol. 28, no. 4, 2017. [Online]. Available: http://doi.org/f3tbpn

[59] 5G Americas, "Management Orchestration & Automation - White Paper," 5G Americas, Tech. Rep., Nov. 2019. [Online]. Available: www.5gamericas.org/wp-content/uploads/2019/11/Management-Orchestration-and-Automation_clean.pdf

[60] F. Z. Yousaf, M. Bredel, S. Schaller, and F. Schneider, "NFV and SDN-Key technology enablers for 5G networks," *IEEE Journal on Selected Areas in Communications*, vol. 35, no. 11, pp. 2468–2478, Nov. 2017. [Online]. Available: https://doi.org/dv2z

[61] ETSI, "GS NFV-MAN 001 - V1.1.1 - Network Functions Virtualisation (NFV); Management and Orchestration," European Telecommunications Standards Institute, Tech. Rep., Dec. 2014. [Online]. Available: www.etsi.org/standards

[62] K. Martiny, D. Telekom, V. Yanover, Z. Lan, R. Mackenzie, B. Khasnabish, J. Groenendijk, H. Alkanani, J. Golic, and T. Italia, "5G Network and Service Management Including Orchestration v3.14.0," Next Generation Mobile Networks Alliance, Tech. Rep., Mar. 2019. [Online]. Available: www.ngmn.org

[63] J. R. Boyd, "A discourse on winning and losing," in *Destruction and Creation*, 1976.

[64] Orange and Huawei, "White Paper: Future OSS - Providing the Agility to support digital operations transformation of hybrid networks," Orange, S.A. and Huawei Technologies, Ltd., Tech. Rep., 2017. [Online]. Available: www.huawei.com

[65] J. Strassner, *Policy-Based Network Management: Solutions for the Next Generation*, 1st ed., M. Kaufmann, Ed. Elsevier Ltd., 2003.

[66] A. R. Choudhary, "Policy-based network management," *Bell Labs Technical Journal*, vol. 9, no. 1, pp. 19–29, 2004. [Online]. Available: https://doi.org/dpdqxq

[67] J. Ding, *Advances in Network Management*, 1st ed., N. Y. A. Publications, Ed. Auerbach Publications, 2010.

[68] N. F. Saraiva de Sousa, D. A. Lachos Perez, R. V. Rosa, M. A. Santos, and C. Esteve Rothenberg, "Network Service Orchestration: A survey," *Computer Communications*, vol. 142-143, p. 69–94, Jun. 2019. [Online]. Available: http://doi.org/dv2v

[69] Anuta Networks, "A Primer on Network Service Orchestration Contents," Anuta Networks, Tech. Rep., Mar. 2018. [Online]. Available: www.anutanetworks.com/wp-content/uploads/2018/03/network-orchestration-ebook.pdf

[70] ONAP, "White Paper: Open Network Automation Platform - Architecture Overview," Linux Foundation Projects, Tech. Rep., Jul. 2019. [Online]. Available: https://www.onap.org/wp-content/uploads/sites/20/2019/07/ONAP_CaseSolution_Architecture_062519.pdf

[71] A. Kapadia, *ONAP Demystified: Automate Network Services with ONAP*. CreateSpace Independent Publishing Platform, North Charleston, SC, USA, 2018.

[72] Cloudify, "ONAP: Orchestration for Real Results - A Guide to ONAP Architecture and Use Cases," Cloudify, Tech. Rep., Feb. 2018. [Online]. Available: https://cloudify.co

[73] Linux Foundation Projects, "ONAP Architecture - El Alto," 2020, last accessed on 16/05/2020. [Online]. Available: https://docs.onap.org/en/elalto/guides/onap-developer/architecture/onap-architecture.html#id1

[74] ONAP, "ONAP 5G Blueprint Overview," Linux Foundation Projects, Tech. Rep., Jul. 2019. [Online]. Available: www.onap.org

[75] A. Hoban, A. T. Sepulveda, G. G. de Blas, K. Kashalkar, M. Shuttleworth, M. Harper, and R. Velandy, "OSM Release ONE: A Technical Overview," European Telecommunications Standards Institute, Tech. Rep., Oct. 2016, White Paper. [Online]. Available: https://osm.etsi.org/wikipub/index.php/Release_notes_and_whitepapers

[76] Intel, Telenor, A. Labs, Netrounds, and RIFT.io, “White Paper: PoC Demonstrates Automated Assurance and DevOps in Service Chains and 5G Network Slices,” European Telecommunications Standards Institute, Tech. Rep., 2017. [Online]. Available: https://osm.etsi.org/wikipub/images/3/3c/PoC_1_White_Paper.pdf

[77] ETSI OSM, “OSM Scope and Functionality - OSM Public Wiki,” 2020, last accessed on 03/05/2020. [Online]. Available: https://osm.etsi.org/wikipub/index.php/OSM_Scope_and_Functionality

[78] ETSI OSM, “OSM Integration Guidelines - OSM Public Wiki,” 2020, last accessed on 03/05/2020. [Online]. Available: https://osm.etsi.org/wikipub/index.php/OSM_Integration_Guidelines

[79] A. Israel, A. T. Sepúlveda, A. Reid, F. Vicens, F. J. R. Salguero, G. G. de Blas, G. Lavado, M. Shuttleworth, M. Harper, M. Marchetti, R. Vilalta, S. Almagia, and V. Little, “OSM Release FIVE: Technical Overview,” European Telecommunications Standards Institute, Tech. Rep., Jan. 2019, White Paper. [Online]. Available: https://osm.etsi.org/wikipub/index.php/Release_notes_and_whitepapers

[80] OpenStack Foundation, “Heat - OpenStack,” p. 1, 2020, last accessed on 04/05/2020. [Online]. Available: https://wiki.openstack.org/wiki/Heat

[81] Fujitsu, “Overview of Heat,” last accessed on 30/04/2020. [Online]. Available: https://doc.cloud.global.fujitsu.com/lib/iaas/en/heat-template/concept/overview_of_heat.html

[82] OpenStack Foundation, “Heat Orchestration Template Guide,” p. 1, 2020, last accessed on 04/05/2020. [Online]. Available: https://docs.openstack.org/heat/latest/template_guide/hot_guide.html

[83] OASIS, “TOSCA Version 1.0,” Nov. 2013, last accessed on 04/05/2020. [Online]. Available: https://docs.oasis-open.org/tosca/TOSCA/v1.0/TOSCA-v1.0.pdf

[84] OASIS, “TOSCA Version 2.0,” Jun. 2020, last accessed on 04/07/2020. [Online]. Available: https://docs.oasis-open.org/tosca/TOSCA/v2.0/TOSCA-v2.0.pdf

[85] M. Owen and J. Raj, “BPMN: Introduction to the New Business Process Modeling Standard,” Popkin Software, Tech. Rep., Mar. 2003. [Online]. Available: www.omg.org

[86] M. Von Rosing, H. Von Scheel, and A. W. Scheer, *The Complete Business Process Handbook: Body of Knowledge from Process Modeling to BPM.* Elsevier Inc., 2014, vol. 1. [Online]. Available: http://doi.org/dv27

[87] S. A. White, “Introduction to BPMN,” IBM Corporation, Tech. Rep., Mar. 2020. [Online]. Available: www.omg.org/bpmn/Documents/Introduction_to_BPMN.pdf

[88] OMG, “Business Process Model and Notation (BPMN), Version 2.0.2,” Object Management Group, Tech. Rep., Dec. 2013. [Online]. Available: www.omg.org/spec/BPMN/2.0.2

[89] Camunda Services GmbH, “Camunda BPM documentation - Introduction,” 2020. [Online]. Available: https://docs.camunda.org/manual/7.12/introduction

[90] M. Dumas, M. La Rosa, J. Mendling, and H. A. Reijers, *Fundamentals of Business Process Management.* Springer Berlin Heidelberg, 2013. [Online]. Available: http://doi.org/b79v

[91] Camunda Services GmbH, "Camunda Best Practices," last accessed on 02/05/2020. [Online]. Available: https://camunda.com/best-practices/_book/

[92] Camunda Services GmbH, "Camunda BPM documentation - Architecture Overview," last accessed on 01/05/2020. [Online]. Available: https://docs.camunda.org/manual/7.12/introduction/architecture

[93] Camunda Services GmbH, "Camunda BPM documentation - Manual," last accessed on 01/05/2020. [Online]. Available: https://docs.camunda.org/manual/7.12

[94] Camunda Services GmbH, "BPMN 2.0 Symbol Reference," last accessed on 01/05/2020. [Online]. Available: https://camunda.com/bpmn/reference

[95] Camunda Services GmbH, "BPMN 2.0 Implementation Reference," last accessed on 01/05/2020. [Online]. Available: https://docs.camunda.org/manual/7.12/reference/bpmn20

[96] Oracle, "Oracle Business Process Management Suite," Oracle Corporation, Tech. Rep., 2014. [Online]. Available: www.oracle.com/technetwork/middleware/bpm/overview/bpm-suite-12c-ds-2264242.pdf

[97] Oracle, "Oracle Business Process Management," last accessed on 01/05/2020. [Online]. Available: www.oracle.com/middleware/technologies/bpm.html

[98] Cisco Systems, "Cisco Annual Internet Report (2018–2023) White Paper," p. 35, 2020, last accessed on 07/06/2020. [Online]. Available: www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html

[99] Altice Labs, S.A., Altran Portugal, S.A., IT, IT Center, Nokia, Onesource, PDM&FC, Ubiwhere, Universidade de Coimbra, "Scenario definition for different traffic profiles," 5GO consortium, Tech. Rep., Jun. 2018, D4.1 - V1.0. [Online]. Available: https://5go.pt/en/results

[100] Z. Wu, J. Zhang, W. Xie, and F. Yang, "CDN Convergence Based on Multi-Access Edge Computing," in *2018* 10th *International Conference on Wireless Communications and Signal Processing, WCSP 2018.* Institute of Electrical and Electronics Engineers Inc., Nov. 2018. [Online]. Available: http://doi.org/d68w

[101] A. Pathan and R. Buyya, "A taxonomy and survey of content delivery networks," *Grid Computing and Distributed Systems Laboratory*, 2007. [Online]. Available: http://www.hit.bme.hu/~jakab/edu/litr/CDN/CDN-Taxonomy.pdf

[102] P. Frangoudis, L. Yala, and A. Ksentini, "CDN-as-a-Service Provision over a Telecom Operator's Cloud," *IEEE Transactions on Network and Service Management*, vol. 14, no. 3, pp. 702–716, Sep. 2017. [Online]. Available: http://doi.org/gbww5b

[103] P. T. Endo, J. Byrne, T. Lynn, R. Loomba, R. Quinn, C. K. Filelis-Papadopoulos, K. M. Giannoutakis, G. A. Gravvanis, D. Tzovaras, P. Willis, and S. Svorobej, "Analyzing resource distribution over a real-world large-scale virtual content infrastructure," in *Proceedings - International Symposium on Computers and Communications*, vol. 2019-June. Institute of Electrical and Electronics Engineers Inc., Jun. 2019. [Online]. Available: https://doi.org/d7t7

[104] C. K. Filelis-Papadopoulos, K. M. Giannoutakis, G. A. Gravvanis, P. T. Endo, D. Tzovaras, S. Svorobej, and T. Lynn, "Simulating large vCDN networks: A parallel approach," *Simulation Modelling Practice and Theory*, vol. 92, pp. 100–114, Apr. 2019. [Online]. Available: https://doi.org/d7tz

[105] ETSI, "GR NFV 001 - V1.2.1 - Network Functions Virtualisation (NFV); Use Cases," European Telecommunications Standards Institute, Tech. Rep., May. 2017. [Online]. Available: www.etsi.org/standards

[106] B. Zolfaghari, G. Srivastava, S. Roy, H. R. Nemati, F. Afghah, T. Koshiba, A. Razi, K. Bibak, P. Mitra, and B. K. Rai, "Content Delivery Networks: State of the Art, Trends, and Future Roadmap," *ACM Comput. Surv.*, vol. 53, no. 2, Apr. 2020. [Online]. Available: https://doi.org/d7vn

[107] Linux Foundation Projects, "Service Design and Creation Project," 2020, last accessed on 09/07/2020. [Online]. Available: https://wiki.onap.org/display/DW/Service+Design+and+Creation+%28SDC%29+Portal

[108] Linux Foundation Projects - ONAP, "ARCHCOM: InfoFlow - SDC Service Distribution," 2019, last accessed on 15/07/2020. [Online]. Available: https://wiki.onap.org/display/DW/ARCHCOM%3A+InfoFlow+-+SDC+Service+Distribution

[109] Linux Foundation Projects, "Active and Available Inventory Project," 2019, last accessed on 15/07/2020. [Online]. Available: https://wiki.onap.org/display/DW/Active+and+Available+Inventory+Project

[110] Linux Foundation Projects, "Service Orchestrator (SO) Project," 2018, last accessed on 25/07/2020. [Online]. Available: https://wiki.onap.org/pages/viewpage.action?pageId=4719246

[111] Linux Foundation Projects, "Virtual Infrastructure Deployment Project," 2018, last accessed on 26/07/2020. [Online]. Available: https://wiki.onap.org/display/DW/Virtual+Infrastructure+Deployment+Project

[112] Linux Foundation Projects, "ONAP El Alto - SO - Architecture — Functional View," 2020, last accessed on 28/05/2020. [Online]. Available: https://docs.onap.org/en/elalto/submodules/so.git/docs/

[113] Linux Foundation Projects, "Virtual Function Controller Project," 2020, last accessed on 22/07/2020. [Online]. Available: https://wiki.onap.org/display/DW/Virtual+Function+Controller+Project

[114] Linux Foundation Projects, “Virtual Function Controller Architecture,” 2020, last accessed on 22/07/2020. [Online]. Available: https://docs.onap.org/en/elalto/submodules/vfc/nfvo/lcm.git/docs/

[115] Linux Foundation Projects, “Multi VIM/Cloud Project,” 2018, last accessed on 23/07/2020. [Online]. Available: https://wiki.onap.org/pages/viewpage.action?pageId=3247262

[116] Linux Foundation Projects, “Microservices Bus Project,” 2018, last accessed on 18/07/2020. [Online]. Available: https://wiki.onap.org/display/DW/Microservices+Bus+Project

[117] Linux Foundation Projects, “Data Movement as a Platform Project,” 2019, last accessed on 16/07/2020. [Online]. Available: https://wiki.onap.org/display/DW/Data+Movement+as+a+Platform+Project

[118] L. Deng, H. Deng, and S. Terrill, “Harmonizing Open Source and Standards: The Progress of ONAP,” Linux Foundation Projects, Tech. Rep., Apr. 2019. [Online]. Available: www.onap.org

[119] Linux Foundation Projects, “External API/NBI - Offered APIs,” 2020, last accessed on 29/05/2020. [Online]. Available: https://docs.onap.org/en/elalto/submodules/externalapi/nbi.git/docs/

[120] Linux Foundation Projects, “ONAP External API Framework Project,” 2019, last accessed on 30/05/2020. [Online]. Available: https://wiki.onap.org/display/DW/External+API+Framework+Project

[121] TeleManagement Forum (TM Forum), “TMF633 ServiceCatalog API,” last accessed on 30/05/2020. [Online]. Available: https://github.com/tmforum-apis/TMF633_ServiceCatalog

[122] TeleManagement Forum (TM Forum), “TMF638 ServiceInventory API,” last accessed on 30/05/2020. [Online]. Available: https://github.com/tmforum-apis/TMF638_ServiceInventory

[123] TeleManagement Forum (TM Forum), “TMF641 ServiceOrder API,” last accessed on 30/05/2020. [Online]. Available: https://github.com/tmforum-apis/TMF641_ServiceOrder

[124] ETSI, “GS NFV-IFA 009 V1.1.1 - Network Functions Virtualisation (NFV); Management and Orchestration; Report on Architectural Options,” European Telecommunications Standards Institute, Tech. Rep., Jul. 2016. [Online]. Available: www.etsi.org/standards

[125] Linux Foundation Projects, “ETSI Alignment Support,” 2020, last accessed on 16/07/2020. [Online]. Available: https://wiki.onap.org/display/DW/ETSI+Alignment+Support

[126] ETSI, “GS NFV-SOL 005 V2.8.1 - Network Functions Virtualisation (NFV) Release 2; Protocols and Data Models; RESTful protocols specification for the Os-Ma-nfvo Reference Point,” European Telecommunications Standards Institute, Tech. Rep., Sep. 2020. [Online]. Available: www.etsi.org/standards

[127] ETSI, "GS NFV-SOL 003 V3.3.1 - Network Functions Virtualisation (NFV) Release 3; Protocols and Data Models; RESTful protocols specification for the Or-Vnfm Reference Point," European Telecommunications Standards Institute, Tech. Rep., Aug. 2020. [Online]. Available: www.etsi.org/standards

[128] ETSI, "GS NFV-SOL 002 V3.3.1 - Network Functions Virtualisation (NFV) Release 3; Protocols and Data Models; RESTful protocols specification for the Ve-Vnfm Reference Point," European Telecommunications Standards Institute, Tech. Rep., Aug. 2020. [Online]. Available: www.etsi.org/standards

[129] Linux Foundation Projects, "ONAP Operations Manager Project," 2020, last accessed on 02/09/2020. [Online]. Available: https://wiki.onap.org/display/DW/ONAP+Operations+Manager+Project

[130] ONAP, "White Paper: ONAP Solution Brief," Linux Foundation Projects, Tech. Rep., Jun. 2018, last accessed on 19/09/2020. [Online]. Available: www.onap.org/wp-content/uploads/sites/20/2018/06/ONAP_CaseSolution_OOM_0618FNL.pdf

[131] L. Bonati, M. Polese, S. D'Oro, S. Basagni, and T. Melodia, "Open, Programmable, and Virtualized 5G Networks: State-of-the-Art and the Road Ahead," *Computer Networks*, vol. 182, p. 107516, May. 2020. [Online]. Available: https://doi.org/d8tm

[132] Linux Foundation Projects, "Setting Up ONAP - El Alto," 2020, last accessed on 03/09/2020. [Online]. Available: https://docs.onap.org/en/elalto/submodules/oom.git/docs/oom_cloud_setup_guide.html

[133] Linux Foundation Projects, "OOM Quick Start Guide," 2020, last accessed on 04/09/2020. [Online]. Available: https://docs.onap.org/en/elalto/submodules/oom.git/docs/oom_quickstart_guide.html

[134] Linux Foundation Projects, "ONAP Design Service," 2020, last accessed on 06/07/2020. [Online]. Available: https://docs.onap.org/en/elalto/guides/onap-user/design/index.html

[135] J. Aires, P. Barbosa, S. Figueiredo, B. Parreira, J. Mamede, and M. Ricardo, "Addressing end-to-end Orchestration of Virtualized Telco Services using ONAP in a R&D environment," *InForum Braga 2019: Comunicações e Redes de Computadores*, p. 12, Sep. 2019.

[136] Linux Foundation Projects, "Service Instantiation via ONAP NBI API (TM Forum)," 2019, last accessed on 17/08/2020. [Online]. Available: https://docs.onap.org/en/elalto/submodules/so.git/docs/developer_info/instantiate/instantiation/nbi/index.html

[137] Linux Foundation Projects, "API consumed by SO - VFC APIs," 2020, last accessed on 20/09/2020. [Online]. Available: https://docs.onap.org/en/elalto/submodules/so.git/docs/api/apis/consumed-apis.html#vfc-apis