João Pedro
Simões Alegria

Orquestração multi-domínio para verticais em serviços de redes 5G

Inter-domain orchestration for verticals in 5G network services

João Pedro
Simões Alegria

Orquestração multi-domínio para verticais em serviços de redes 5G

Inter-domain orchestration for verticals in 5G network services

"*Failure is the opportunity to begin again more intelligently.*"

— Henry Ford

Universidade de Aveiro
2021

João Pedro
Simões Alegria

Orquestração multi-domínio para verticais em serviços de redes 5G

Inter-domain orchestration for verticals in 5G network services

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia Informática, realizada sob a orientação científica do Doutor Diogo Nuno Pereira Gomes, Professor auxiliar do da Universidade de Aveiro, e do Doutor Daniel Nunes Corujo, Professor auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro.

Dedico este trabalho à minha família, amigos e namorada, que desde sempre estiveram ao meu lado e me apoiaram incansavelmente.

o júri / the jury

presidente / president

Prof. Doutor Ilídio Fernando de Castro Oliveira
professor auxiliar da Universidade de Aveiro

vogais / examiners committee

Prof. Doutor Carlos Jesus Bernardos Cano
professor associado da Universidade Carlos III de Madrid

Prof. Doutor Diogo Nuno Pereira Gomes
professor auxiliar da Universidade de Aveiro

agradecimentos / acknowledgements

Muito obrigado aos meus orientadores, não só pelo apoio, mas também pela oportunidade apresentada.Agradeço também a todos os meus amigos que me ajudaram sempre que necessário, em particular a todos os colegas do Instituto de Telecomunicações de Aveiro a trabalhar no projeto 5Growth, que me guiaram e ajudaram a compreender os aspetos complexos que esta dissertação aborda. Queria também agradecer aos meus colegas de curso, Filipe Pires e André Pedrosa, pela ajuda e paciência. Agradecimento ao Instituto de Telecomunicações de Aveiro e ao projeto EC H2020 5GPPP 5Growth project (Grant 856709).

Palavras Chave 5G, NFV, MANO, virtualização, multi-domínio, orquestração

Resumo

Com a proliferação das tecnologias 5G, tanto as operadoras como as verticais estão cada vez mais interessadas em utilizar as suas capacidades. Ao suportar tais tecnologias, as operadoras deixam de usar hardware especializado de rede, como routers e switches, e começam a virtualizar as suas funções, usando servidores comercialmente disponíveis para executar os seus serviços e redes. Essa virtualização reduz os custos de associados e aumenta o número de cenários suportados sem precisar de alterar a infraestrutura. Por outro lado, as verticais estão cada vez mais interessadas nas garantias de desempenho que as tecnologias 5G oferecem. Com os desenvolvimentos nesta área e o aparecimento de cenários mais complexos, o uso de serviços de rede multi-domínio aparece como uma solução para alguns desses casos de uso. Atualmente, as abordagens utilizadas para implementar o mecanismo multi-domínio são restritivas e principalmente geridas pelas operadoras, o que precisa de ser equilibrado, dando também algum poder às verticais. Esta tese visa abordar os problemas atuais na orquestração de serviços de rede 5G em cenários multi-domínio para verticais, criando uma nova plataforma de orquestração chamada Network Orchestrator (NetOr).

Depois de analisar as soluções existentes para a orquestração de serviços verticais e identificar os seus problemas, conclui que o NetOr deve resolver as arquiteturas monolíticas, o slicing de rede não standardizado e o suporte incompleto do mecanismo multi-domínio. Depois de desenvolver o sistema, ele foi testado extensivamente com diferentes tipos de testes, o que exigiu a criação de diversas entidades e estruturas de configuração, tais como VNFs, NSDs e NSTs. Esses testes incluíram testes de unidade, testes funcionais e testes de desempenho, comparando os valores obtidos com os da solução mais madura de orquestração de serviços verticais atualmente existente. Esses testes concentraram-se no uso do mecanismo multi-domínio integrado no novo sistema NetOr, aplicado a um cenário de slicing de rede. Os resultados obtidos nesses testes provaram que o NetOr é um sistema promissor e similar em termos de desempenho às melhores soluções de orquestração de serviços verticais disponíveis. No futuro, desenvolvimentos e melhorias devem ser feitas sobre o sistema para que ele se torne mais competitivo na arena de orquestração de serviços de rede verticais.

Keywords 5G, NFV, MANO, virtualization, inter-domain, orchestration

Abstract

With the proliferation of 5G technologies, both operators and verticals are increasingly interested in using its capabilities. By supporting these technologies, operators can stop using specialized networking hardware, such as routers and switches, and start virtualizing their functions, using commercially available servers to run their services and networks. Virtualization considerably reduces costs while increasing the number of scenarios supported without changing the infrastructure. On the other hand, verticals are increasingly interested in the performance guarantees the 5G technologies provide. With the developments in this area and the appearance of more complex scenarios, the use of inter-domain network services appears as a solution for some use-cases. Currently, the approaches used to implement the inter-domain mechanism are restrictive and mainly managed by operators, which needs to be balanced by giving some power to the verticals. This thesis' work addresses the current problems in inter-domain 5G network services orchestration for verticals by creating a new orchestration platform called Network Orchestrator(NetOr).

After analyzing the state-of-the-art solutions and identifying their problems, I concluded that the NetOr should solve the monolithic architectures, the non-standardized network slicing, and the incomplete inter-domain support. After developing the system, it was extensively tested with different approaches, which demanded the creation of many entities and configuration structures, such as VNFs, NSDs, and NSTs. Those tests included unit tests, functional tests, and performance tests, comparing the gathered values with the most mature state-of-the-art vertical service orchestration solution. These tests focused on using the inter-domain mechanism integrated into the new NetOr system in a network slicing scenario. The results obtained from those tests proved that the NetOr is a promising platform, being comparable to the best available vertical service orchestration solutions. Further future developments over the NetOr system are needed to make it a more competitive solution in the vertical network service orchestration arena.

Contents

List of Figures

List of Tables

Glossary

5G	Fifth Generation
4G	Fourth Generation
VS	Vertical Slicer
SO	Service Orchestrator
OSM	Open Source MANO
ETSI	European Telecommunications Standards Institute
MANO	Management and Orchestration
E2E	End-to-End
SDN	Software-Defined Networks
NFV	Network Functions Virtualization
NBI	Northbound Interface
SBI	Southbound Interface
E/WBI	Eastbound/Westbound Interface
VIM	Virtual Infrastructure Manager
VNF	Virtual Network Function
PNF	Physical Network Function
VM	Virtual Machine
NS	Network Service
COTS	Commercially Over-The-Shelf
NFVI	Network Functions Virtualization Infrastructure
VNFFG	Virtual Network Function Forwarding Graph
VL	Virtual Link
NFVO	Network Function Virtualization Orchestrator
VNFM	Virtual Network Function Manager
OSS	Operations Support System
BSS	Bussiness Support System
VNFD	Virtual Network Function Descriptor
PNFD	Physical Network Function Descriptor
WIM	WAN Infrastructure Manager
NSD	Network Service Descriptor
VLD	Virtual Link Descriptor
VNFFGD	VNF Forwarding Graph Descriptor
FCAPS	Fault Management, Configuration Management, Accounting Management, Performance Management, and Security Management
EM	Element Management
NSI	Network Slice Instance
NST	Network Slice Template
CN	Core Network
AN	Access Network
NSSI	Network Slice Subnet Instance
RAN	Radio Access Network
MTP	Mobile Transport and Computing Platform
5GT	5G-Transformer
QoS	Quality of Service
VSB	Vertical Service Blueprint
VSD	Vertical Service Descriptor
VSI	Vertical Service Instance
MVNO	Mobile Virtual Network Provider
SLA	Service Level Agreement
LCM	LifeCycle Manager
NSMF	Network Slice Management Function
NSSMF	Network Slice Subnet Management Function
PoP	Point of Presence
FIRE	Future Internet Research and Experimentation
AAA	Authentication, Authorization and Accounting
IdP	Identity Provider
RFS	Resource Facing Service
CFS	Customer Facing Service
AI	Artificial Intelligence
eMBB	enhanced Mobile BroadBand
URLLC	Ultra-Reliable Low-Latency Communication
mMTC	massive Machine Type Communication
5GR	5Growth
RL	Resource Layer
ML	Machine Learning
CD	Continuous Delivery
CI	Continuous Integration
CSMF	Communication Service Management Function
NSMF	Network Service Management Function
3GPP	3rd Generation Partnership Project
MTD	Moving Target Defense

CMD	Cyber Mimic Defense
MEC	Multi-access Edge Computing
PPP	Public Private Partnership
CAPEX	Capital Expenses
OPEX	Operational Expenses
SoA	State of the Art
CRUD	Create, Read, Update and Delete
REST	REpresentational State Transfer
ELK	Elastic Search, Logstash and Kibana
CSP	Communication Service Providers
NR	New Radio
NG	New Generation
NRM	New Radio Model
API	Application Programmable Interface
NetOr	Network Orchestrator
VPN	Virtual Private Network
VDU	Virtual Deployment Unit
URL	Uniform Resource Locator
IP	Internet Protocol
OS	Operating System
TICK	Telegraf, InfluxDB, Chronograf and Kapacitor
POC	Proof-of-Concept
KPI	Key Performance Indicators
SLO	Service Level Objectives
SDO	Standards Development Organization
DSP	Digital Service Provider
DSC	Digital Service Consumer
NSP	Network Service Provider
ONAP	Open Network Automation Platform
AWS	Amazon Web Services
TOSCA	Topology and Orchestration Specification for Cloud Applications
YAML	YAML Ain't Markup Language
MNO	Mobile Network Operator
vMNO	virtual Mobile Network Operator
DBMS	Database Management System
NF	Network Function

CHAPTER 1

Introduction

1.1 Motivation

With the continuous technological advancements, the computational resources becoming increasingly faster, and the decrease of computer hardware cost, having a powerful machine is significantly easier. To a degree, the new Fifth Generation (5G) technologies use that evolution. Besides being the new generation of radio networks, it also has core concepts such as Network Functions Virtualization (NFV), Software-Defined Networks (SDN), and End-to-End (E2E) Network Slicing that favor the virtualization of specialized network hardware (routers and switches), replacing it with Virtual Machines (VMs) with specific software to serve equivalent functions and services. By adopting this virtualization, the new VMs can be deployed in Commercially Over-The-Shelf (COTS) hardware, decreasing the infrastructure's cost while increasing the number of scenarios supported without changing the infrastructure. This approach also allows the reduction of Capital Expenses (CAPEX) since the commercially available hardware is cheaper, and Operational Expenses (OPEX) because that hardware can be maintained or replaced with minimal cost.

Knowing all those advantages and possibilities, both verticals and network/service providers want to use those new 5G technologies. Operators want to support those functionalities due to the decrease of expenses in having and maintaining the network infrastructure. Verticals and service clients are interested in the 5G technologies because they open many opportunities while guaranteeing performance improvements. With the evolution in this area and the efforts to support and integrate such technologies, scenarios, and services with higher complexity appear. Some of the more intricate and discussed are the inter-domain scenarios, where the E2E service span multiple administrative domains, increasing the service's coverage area. Solutions and systems supporting these scenarios already exist, but their support is still in an initial phase, mainly using a federation-like approach. That is not an inter-domain environment, since federation implies a pre-defined connection between domains.

A new inter-domain approach is required to deploy a fully-fledged E2E inter-domain vertical service across different administrative domains without prior knowledge of each other.

This mechanism heavily relies on the network slicing capabilities of segmenting the entire network, ranging from Radio Access Network (RAN), transportation networks, and Core Network (CN). The network slice is also the entity that composes and manages different network services or other network slices, an aspect needed in this scenario.

1.2 Objectives

This thesis aims at addressing the current problems in inter-domain 5G network services orchestration for verticals. Developments in this area increasingly demand the support of network slicing, a functionality that is adopted in different ways by State of the Art (SoA) projects (systems providing a version of multi-domain service orchestration to industry verticals). With the implementation of network slicing, those systems allow the segmentation of the entire infrastructure, ranging from network to computational resources. The created vertical slices are isolated and strictly follow the Quality of Service (QoS) requirements established by verticals, allowing different service types within the same infrastructure.

The thesis work goal is to address the recurrent problems of SoA solutions by developing a new and improved vertical service orchestration platform. The issues that will be solved are:

- the monolithic architecture adopted by all vertical service orchestration platforms;
- the non-standardized network slicing support;
- the incomplete inter-domain support.

This Proof-of-Concept (POC) aims at validating the proposed solutions and verify if the defined architecture is advantageous for vertical service orchestration. At the same time, it also validates the new proposed approach to the inter-domain service orchestration based on network slices.

1.3 Thesis Structure

The thesis is composed by five chapters and two appendix. Those chapters are:

- Chapter 1: Introduction
- Chapter 2: 5G Technology
- Chapter 3: Vertical Service Orchestrators
- Chapter 4: Proof of Concept
- Chapter 5: Results
- Chapter 6: Conclusions

Chapter 1 presents the motivation, objectives, and structure of this thesis. Chapter 2 briefly contextualizes the 5G technologies, its core concepts and frameworks. This chapter also presents the state-of-the-art approaches when using these 5G technologies, defined in various standards and scientific works. Finally, it presents the current orchestration challenges and solutions. Chapter 3 presents the most relevant vertical service orchestration platforms, their architecture, benefits, and disadvantages. Chapter 4 presents the Network

Orchestrator (NetOr) POC system architecture, detailing every component, major system features, and deployment approach. This chapter also details the implementation hardships experienced during the platform's development phase. Chapter 5 describes all the tests and results obtained from this new system, providing a discussion and explanation of all gathered values. Finally, Chapter 6 presents the conclusions obtained from the development of this system, how it compares to existing solutions, and future work considered. Appendix A contains screenshots of the main pages for both the admin and tenant profiles of the web portal developed for the NetOr system. Appendix B contains the onboarded Vertical Service Blueprint (VSB), Vertical Service Descriptor (VSD), and Vertical Service Instance (VSI) data models used during the tests performed.

CHAPTER 2

5G Technologies

To give some contextualization, 5G is the new generation of wireless network technology, which will enable the connection of much more devices, each with a much faster network connection than the existing Fourth Generation (4G) ones. Furthermore, 5G also considers the wide range of needs and requirements from vertical industries (businesses focused on supplying specialized services and products to each other[1]). According to [2], from the technology's commercial launch in 2019, 62 operators launched their 5G services in 32 markets, and 94 operators announced their plans to launch their own just in one year. Estimations suggest that in 2025 20% of the world's customer base will account for 5G subscriptions, the reasoning behind operators investing upwards of one trillion Euros in mobile infrastructures, where 80% are 5G. Although offered as a consumer-facing technology applied to gaming scenarios, this technology has already been validated and evaluated for industrial applications in more than 180 trials across 28 European member states. Verticals want to use and improve these 5G technologies, mainly because of the new business opportunities they bring and the potential to fulfill network constraints, such as bandwidth and latency.

Several organizations facilitate developments in new 5G solutions and standardize core concepts. Some of those organizations are 5G-Public Private Partnership (PPP)[1], 3rd Generation Partnership Project (3GPP) [2], and European Telecommunications Standards Institute (ETSI) [3]. 5G-PPP is "a joint initiative between the European Commission and European ICT industry (ICT manufacturers, telecommunications operators, service providers, SMEs and researcher Institutions)"[3], which deliverers solutions, architectures, technologies, and standards for the upcoming 5G infrastructures. 3GPP is a partnership between seven other telecommunications Standards Development Organization (ARIB[4], ATIS[5], CCSA[6], ETSI,

[1] https://5g-ppp.eu/
[2] https://www.3gpp.org/
[3] https://www.etsi.org/
[4] www.arib.or.jp
[5] www.atis.org
[6] www.ccsa.org.cn

TSDSI[7], TTA[8], TTC[9]) that provides radio access, core network, service capability guidelines, and standards through publishing reports and specifications. ETSI is an organization dealing with telecommunications, broadcasting, other electronic communications networks, and services standardization, being the reference organization in Europe.

The core concepts of this new network technologies are Service-based Architecture, SDN, NFV, and E2E Network Slicing [3].

2.1 Service-Based Architecture

A service-based architecture is crucial in this context since, similarly to the service-oriented architecture already used in software development, the division of applications in their constituting services allows the usage of services from different providers. Given they follow the same interface, it is even possible to replace services, the reason for the standardizing efforts.

2.2 SDN

SDN is a fundamental aspect in 5G since it decouples the application, control, and infrastructure layers of existing networks. The SDN architecture defines that there should exist a centralized agent, composed of one or more SDN controllers, with the knowledge of the entire network (topology and status), allowing the definition of the best path to forward the traffic. Comparing this approach to the static architectures of traditional networks, SDN allows much more flexibility by programmatically configuring the network. SDN controllers that constitute the network control plane expose both an Northbound Interface (NBI) and an Southbound Interface (SBI). The NBI is used to connect to network applications present in the management plane, and the SBI is used to connect to the network infrastructure, typically using the OpenFlow protocol.

During a project lifetime, the company responsible for it will undergo two types of expenses: CAPEX, which consist of the funds used by a company or project to acquire, upgrade, and maintain physical assets such as property, buildings, technology, or equipment; and OPEX, which are the expenses a business encounters through its normal business operations, such as rent, equipment, inventory costs, marketing, etc. By adopting these SDN concepts and technologies, CAPEX will decrease since the acquisition and maintenance of highly specialized expensive networking hardware is no longer needed. Similarly, OPEX will also lower since there is no maintenance of physical hardware or the property that hosts it.

Some of the currently available options to implement these kinds of systems are **OpenDayLight**[10], an open-source collaborative project led by the Linux Foundation; **ONOS**[11],

[7] http://tsdsi.org/
[8] www.tta.or.kr
[9] www.ttc.or.jp
[10] https://www.opendaylight.org/
[11] https://wiki.onosproject.org/

also an open-source project headed by the Linux Foundation; **Project FloodLight**[12], a Java-based Apache-licensed open source project; and **Beacon** [13], another Java SDN project.

2.3 NFV

Network Functions Virtualization consists of replacing specialized network hardware with VMs, which uses hypervisors to run networking software and processes (routing, load balancing, etc.).

The Virtual Network Function (VNF) concept is an abstraction for all the VMs and networks needed to implement a given network function. A VNF also exposes connection points originated from VM interfaces or an internal network, enabling them to connect to other VNFs or network providers [4]. Instead of VMs, there is also the option of using containers to implement a portion or the entirety a VNF. Another abstraction created is the Network Service (NS), which consists of a chain of VNFs or Physical Network Functions (PNFs). This abstraction not only encapsulates all the chained network functions but also takes into consideration the associated Virtual Network Function Forwarding Graphs (VNFFGs) and Virtual Links (VLs).

By using VMs or containers, the new virtualized network functions can run in COTS hardware or generic virtualized environments, stopping the need to buy, configure, and connect dedicated hardware. With this approach, the same result can be achieved by using significantly less time and money. Also, especially if using the referred virtualized environment, the deployment is much more flexible, enabling scaling up in peak times and scaling down when the network traffic is low. NFV also allows network engineers to programmatically add and define network functionalities, such as load balancing, routing, firewall security, and automatic provisioning, all in a much faster, easier, and dynamic fashion.

As shown in Figure 2.1, there are three main components in the NFV architecture:

1. **NFV Infrastructure**: This component considers all the underlying physical resources and how to virtualize them, either using containers or VMs.
2. **Virtual Network Functions**: This component corresponds to the software implementation of a network function capable of running over the Network Functions Virtualization Infrastructure (NFVI).
3. **NFV Management and Orchestration**: This component is responsible for orchestrating and managing the lifecycle of both physical and software resources and the lifecycle of VNFs.

Using NFV, it is unnecessary to host any physical machines, decreasing both CAPEX and OPEX due to the same reasons as presented for SDN. Many times, those costs and responsibilities belong to the NFVI, since it should have and maintain high capacity servers in one or more data centers.

[12] https://floodlight.atlassian.net/wiki/spaces/HOME/overview?mode=global
[13] https://openflow.stanford.edu/display/Beacon/Home.html

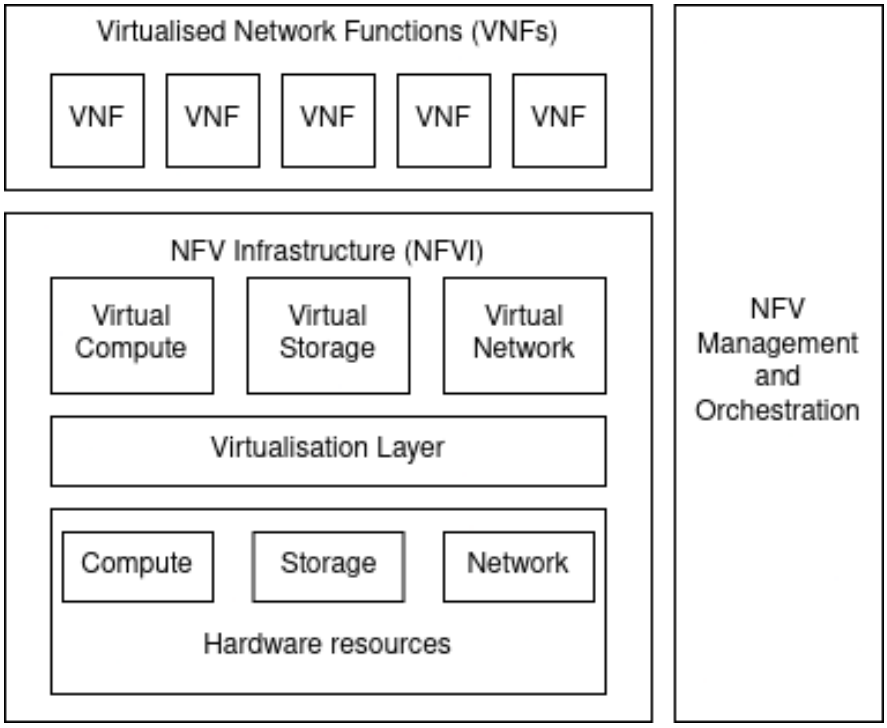


Figure 2.1: High Level NFV Architecture[4]

2.3.1 ETSI NFV-MANO Reference Architectural Framework

Although very powerful, NFV by itself has some limitations. For that reason, ETSI extended its capacities by defining and standardizing an architectural framework that focuses on the changes, functional blocks, and reference points needed for virtualizing operator networks [4]. Figure 2.2 presents the ETSI reference architectural framework, where the most relevant components are the Network Function Virtualization Orchestrator (NFVO), the Virtual Network Function Manager (VNFM), and the Virtual Infrastructure Manager (VIM). The following subsections will briefly explain each entity.

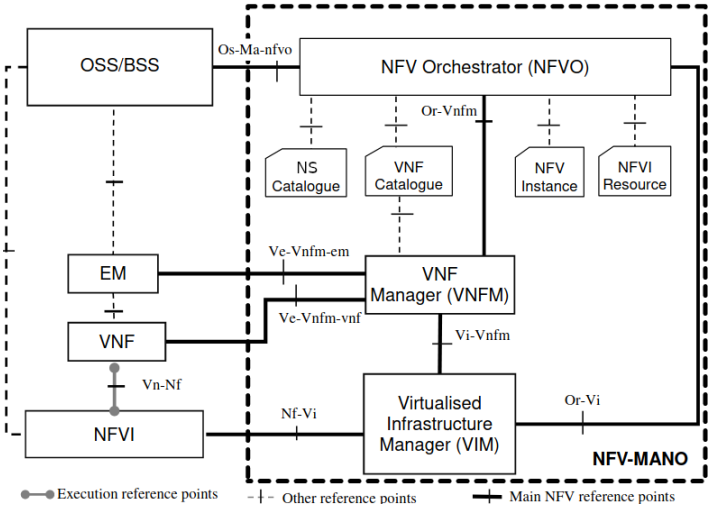


Figure 2.2: ETSI-NFV reference architectural framework [4]

NFVO

The NFVO has two core responsibilities, the orchestration of NFVI resources across multiple VIMs (Resource Orchestration) and the lifecycle management of NSs (Network Service Orchestration). These responsibilities can be kept within the same functional block or separated to enable a multi-vendor deployment or different mappings of functionalities to Administrative Domains [5].

Some of the capabilities provided by the NFVO concerning its Network Service Orchestration functions are:

- Management of NS templates and VNF packages.
- NS instantiation and NS instance lifecycle management.
- Management of the instantiation of VNFMs.
- Management of the instantiation of VNFs.
- Validation and authorization of NFVI resource requests from VNFMs.
- Management of the integrity and visibility of the NS instances.
- Management of the NS instances topology.
- Management of the NS instances automation.
- Policy management and evaluation for the NS instances and VNF instances.

In terms of the other responsibility of the NFVO, the Resource Orchestration functions, some of the functions this component should perform are:

- Validation and authorization of NFVI resource requests from VNFMs.
- NFVI resource management across operator's Infrastructure Domains.
- Supporting the management of the relationship between the VNF instances and the NFVI resources.
- Policy management and enforcement for the NS and VNF instances.
- Collect usage information of NFVI resources by VNF instances or groups of VNF instances.

VNFM

The VNFM is responsible for the lifecycle management of VNF instances. Each VNF should have a manager associated, but each manager can be responsible for one or more VNFs, being or not of the same type. These managers implement generic enough functions able to be applied to any VNF. This NFV-Management and Orchestration (MANO) framework also considers and supports instances that need specific functionalities for their lifecycle management, allowing the VNF package to include those functionalities.

The following set of functionalities is a non-exhaustive set of the functions performed by the Virtual Network Function Manager.

- VNF instantiation and configuration.
- VNF instantiation feasibility checking.
- VNF instance software update/upgrade.
- VNF instance modification.

- VNF instance scaling .
- Collection of VNF instances related metrics.
- VNF instance healing.
- VNF instance termination.
- VNF lifecycle management change notification.s
- VNF integration management.
- Coordination and event reporting between the VIM and the Element Management (EM).

The Virtual Network Function Descriptor (VNFD) standard abstraction captures the necessary configurations, operational behavior, and deployment procedures. This descriptor acts as a recipe containing all the attributes and requirements (such as resource allocation criteria) needed by the MANO platform to create the VNFs and manage their lifecycle. These descriptors should have a one-to-one relationship with VNF packages, the objects stored in a VNF Catalog and always available to be used. When requested, the NFVI will fetch the VNF package from the catalog and allocate the resources indicated on it, guaranteeing it complies with previously defined requirements, constraints, and policies. Some of those policies are "operator policies, geo-location placement, affinity/anti-affinity rules, or local regulations" [5].

VNFDs also increase the deployment flexibility and portability of VNFs, since they enable the instance of VNFs in multiple vendors and diverse NFVI environments (different computing resource generations, multiple virtual network technologies, etc.). Although the use of these descriptors significantly helps, the underlying hardware resources should be well abstracted.

VIM

The VIM has the responsibility of controlling and managing the NFVI's compute, storage, and network resources within one operator's infrastructure domain. Each VIM can manage all those resources or be specialized in a certain one, meaning that it may only provide compute, storage, or networking resources.

Through its NBI, the VIM should support the management of NFVI virtualized compute, storage, and networking resources. On the other hand, its SBI should connect to a variety of hypervisors and network controllers, which together will be the basis of the functionalities exposed by the NBI. Different VIM implementations may expose the interfaces of hypervisors or network controllers as a specialized VIM. An example of this approach is the WAN Infrastructure Manager (WIM), usually used to establish connectivity between PNF endpoints in different NFVI-Point of Presences (PoPs).

The functionalities the VIM should perform are:

- Orchestrating the allocation/upgrade/release/reclamation of NFVI resources.
- Management of the association of the virtualized resources to the physical compute, storage and networking resources.
- Supporting the management of the VNFFGs.
- Management of an information repository about NFVI hardware resources.
- Management of the virtualized resource capacity.
- Management of the software images.

- Collecting performance and fault information from the NFVI.
- Management of catalogues of virtualized resources that can be consumed from NFVI.

From my knowledge, the most relevant VIM options are: **OpenStack**, the most prominent project on Cloud Computing composed of several smaller projects, each one with a given responsibility (compute, storage, network, etc.); **OpenVIM**, an opensource Open Source MANO (OSM) VIM solution that implements the ETSI NFV architecture and allows an all-in-one installation and integration with OSM; and **VMWare**, more specifically VMWare vCloud NFV, a commercially available virtualization solution aligned with ETSI NFV architecture that provides compute, storage and network resources, being broadly used by telecommunication operators.

NS Catalogue

The NS Catalogue is a repository where all the onboarded Network Service Descriptors (NSDs) are stored. NSDs were created to abstract and encapsulate all the necessary VNFs and internal networks for the intended service. Those descriptors aggregate others such as Virtual Link Descriptors (VLDs), VNF Forwarding Graph Descriptors (VNFFGDs) and VNFDs to define and configure the final service.

This catalogue's existence allows the creation and management of NSs deployment templates via the interface operations exposed by the NFVO.

VNF Catalogue

The VNF Catalogue is a repository where all the onboarded VNFDs should be stored. Both the NFVO and the VNFM can query the VNF Catalogue for finding or retrieving VNFDs.

This catalogue's existence allows the creation and management of VNF Packages via the interface operations exposed by the NFVO.

NFV Instances repository

This NFV Instance repository is the one holding all the information related to the VNFs and NSs instances. Several entities may update the NSs records during its lifecycle. Similarly, several entities may also update the VNFs records during its lifecycle. As stated in [5], "this supports the NFVO's and VNFM's responsibilities in maintaining the integrity and visibility of the NS instances, respective VNF instances, and the relationship between them".

NFVI Resources repository

The NFVI Resources repository holds all the information related to the available/reserved/allocated resources from the VIMs in the different operator's infrastructure domains. This repository is crucial because it contains information related to resource reservation, allocation, and monitoring. It also enables the NFVO's Resource Orchestration and governance actions by correlating the NFVI reserved/allocated resources with NS and VNF instances.

Element Management

The EM is responsible for the "Fault Management, Configuration Management, Accounting Management, Performance Management, and Security Management (FCAPS) management functionality for a VNF"[4]. The EM can be aware of virtualization and collaborate with the VNFM to perform the functions that may require information exchange concerning the NFVI resources.

The functions the EM performs may include:

- Configuration for the network functions provided by the VNF.
- Fault management for the network functions provided by the VNF.
- Accounting for the usage of VNF functions.
- Collecting performance measurement results for the functions provided by the VNF.
- Security management for the VNF functions.

Operations Support System/Business Support System

The Operations Support System (OSS)/Bussiness Support System (BSS) combines the operators' functions and processes related to operations and business not explicitly captured by the framework. Although not captured, they still need to be performed, possibly exchanging data with the MANO platform. Most of the time, those functions have full E2E knowledge of services provided by legacy networks and manage/orchestrate legacy systems.

Network Functions Virtualisation Infrastructure

In the MANO environment, the NFVI encapsulates all the hardware (compute, storage and networking) and the software (hypervisors) components that together provide the infrastructure where the VNFs are deployed. On this infrastructure, both fully virtualized network functions and fully physical network functions may be deployed.

Lastly, this infrastructure environment also supports partially virtualized functions, necessary for those with some physical constraints, such as digital interfaces to analog physical channels or vendor design choices. Examples of this phenomenon are hardware load balancers, DSL Access Multiplexers (DSLAMs), Broadband Remote Access Server (BRAS), Wi-Fi access points, CPEs, etc.

2.4 E2E Network Slicing

Network Slicing consists of segmenting the existing monolithic networks and systems in logical partitions. With the appropriate topology and resources, the created isolated segments fulfill a given service category, serve a specific purpose, or even an individual customer. enhanced Mobile BroadBand (eMBB), Ultra-Reliable Low-Latency Communication (URLLC), and massive Machine Type Communication (mMTC) are the three main service categories defined by 3GPP, each with a different set of requirements. 3GPP is the core organization publishing standards and studies in this area since it focuses on radio access and core network concepts, which comprise the whole sliceable network.

A Network Slice Instance (NSI) is an entity managed within the operator's network and with a lifecycle independent of the service instance. A given service is not necessarily active during the entire run-phase duration of its supporting NSI.

These instances can have different completion levels concerning a given business purpose, defined by if the NSI has all the resources and functionalities needed to support all services. Because an NSI comprises one or more network functionalities, it must contain information concerning both functions and their existing interconnections (for example, connections topology, individual requirements, etc.). NSIs also require policies and instance-specific configurations since they can support several service categories and different network portions (CN or Access Network (AN)). Lastly, restricted by its resources (physical and logical), NSIs can be fully, partly, logically, or physically isolated. The created Network Slice Template (NST) concept defines Network Slices deployment guidelines, where the NSI deployment should follow the previously created template and instance-specific information [6].

Figure 2.3 depicts the four phases of the NSI lifecycle:

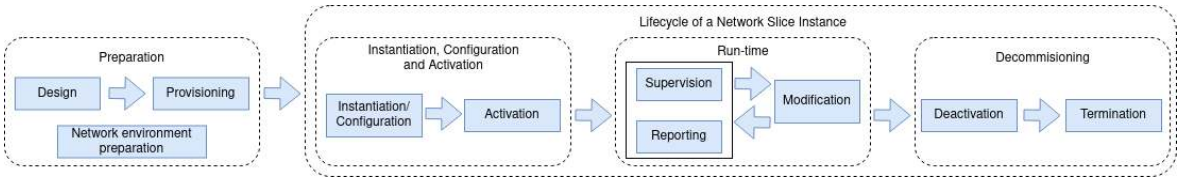


Figure 2.3: NSI lifecycle phases [6]

The output of each phase's task needs appropriate verification. Those tasks are:

- **Preparation phase**: In this phase, the NSI doesn't exist yet. The creation, onboarding, and verification of the NST, the preparation of necessary network environments, or any other type of configurations are the tasks included in this phase.
- **Instantiation, Configuration, and Activation phase**: In this phase, all shared and dedicated resources to the NSI will be created and configured to activate the NSI, including the instantiation, configuration, and activation of shared and non-shared network functions. Some actions performed in the activation step are responsible for effectively activating the NSI, such as redirecting traffic to the instance.
- **Run-time phase**: During this phase, the NSI is capable of handling traffic to support the communication services. Tasks like supervision/reporting for Key Performance Indicators (KPI) reporting and even modifications are the ones expected to occur in this phase. Some of the possible actions are upgrading, reconfiguring, and scaling NSIs, being also possible to associate/remove network functions.
- **Decommissioning phase**: This last stage of the NSI lifecycle includes deactivating the instance, reclaiming dedicated resources, and reconfiguring shared/dependent resources. "After decommissioning, the NSI does not exist anymore" [6].

Subdividing a Network Slice creates Network Slice Subnet Instances (NSSIs), that can significantly help in network management. Taking the example presented in [6], imagining an NSI that contains RAN and CN components, it can be defined and instantiated as two

NSSIs: one responsible for RAN and another responsible for CN. Instantiating both NSSIs combined, the user achieves the intended NSI. Exchanging one of the NSSIs, given that they provide the same functionalities, creates a new NSI.

NSSIs have some interesting characteristics as well: they are composed of network functions or other NSSIs, which in turn can be shared by multiple NSSIs; different NSIs can share the same subnet instance; similarly, multiple NSSIs can share a network function. Finally, NSSIs may contain only CN functions, only AN functions, or both, depending on the purpose.

2.5 State of the Art

The evolution of those technologies interested many people, leading to the study of their main challenges, use cases, opportunities, and managed entities. This work was performed by people from different backgrounds, ranging from researchers to Standards Development Organizations (SDOs), such as ETSI and 3GPP. The most relevant developments happened in the NFV and network slicing areas, focusing on the definition and management of entities like Virtual Network Functions, Network Services, and Network Slices.

After analyzing the works related to network slicing, service orchestration, and management of network entities, I focused on those the most used by the community and accepted as the best when defining and proposing provisioning and management solutions. With that in mind, a set of crucial works was identified, many of them standards. Those are Network Slice Lifecycle Management Model[7], Network Slicing for 5G with SDN/NFV[8], TS 28.531 [9], TS 28.530 [10], TS 28.801 [6], TS 28.541 [11], NFV-EVE 12 [12], SOL005 [13], and SOL006 [14]. The following subsections will briefly describe those studies and their specialized area.

Network Slicing for 5G with SDN/NFV[8]

This paper proposes a SDN/NFV approach to network slicing. The authors of this paper define a network slice as a collection of resources that, appropriately combined, meet the service requirements of the use case that such a slice supports. Those resources may be Network Functions (NFs), blocks that provide specific network capabilities, or infrastructure resources, such as the hardware and software needed to host the NF. According to the authors, those slices are meant to be instantiated through the usage of NFV, and interconnected and exposed to the end-user through the usage of SDN.

For that reason, according to the example presented by the authors, alongside the VNFs needed for the network slice, two additional functions should be instantiated, one serving as a router to connect all VNFs and another serving as the SDN controller managing the slice connectivity.

Finally, the authors present some challenges when implementing and managing network slices in 5G systems, namely the potential performance problems of shared infrastructures, the issues originated by the complex orchestration of slices, and a need to maintain the security and privacy of the system when dealing with network slices.

TS 28.530 [9]

This 3GPP standard studies the network slicing management and orchestration area, defining concepts, use cases, and requirements related to 5G networks and network slices.

The standard starts with an extensive concept and background analysis. That background analysis comprises an overview of the management of 5G networks and networks slicing, the main types of communication services offered by Communication Service Providerss (CSPs), the requirements and examples of communication services using network slices, and lifecycle and concepts related to Network Slices. Additionally, network slicing management aspects are also detailed, both concerning network slices and network subnet slices.

Finally, business-level requirements are presented, followed by a detailed list of high-level use cases, ranging from provisioning actions to management and exposure operations. The document also defines the goal, actors and roles, resources, assumptions, pre-conditions, and steps for all use cases.

TS 28.531 [10]

By specifying guidelines for provisioning network slicing resources, which fall under the management and orchestration operations, this 3GPP specification is possibly one of the most relevant standards for the features and objectives intended for orchestration systems.

The specification starts with a brief overview of the entities related to network slicing, which are Network Slice and Network Slice Subnet, their lifecycle, and their general information model. An explanation of each supported use case, its goal, actors, resources, assumptions, pre-conditions, and steps are also presented, ranging from creation and terminations to the activation, deactivation, modification, and data exposure of both NSIs and NSSIs. Additionally, the document also details use cases related to the creation and configuration of 3GPP networks and sub-networks.

After an overview of management services needed to handle all those scenarios and their mapping, a list of provisioning operations is presented, composed of the following actions: allocateNsi, allocateNssi, deallocatNsi, deallocateNssi, allocateNetwork. Besides, the document also presents a flow chart for how each provisioning action should behave and with which entities it should interact.

Finally, a mapping between the proposed provisioning actions and a RESTful HTTP-based solution is detailed. The operations' names remain the same, being, for example, the allocation action translated in a POST request and the deallocation mapped to a DELETE request. A general description of the information each request should contain is also defined.

TS 28.801 [6]

This 3GPP standard investigates and recommends solutions for management and orchestration operations of network slicing in the Network Slice Instance Layer.

The document starts by introducing the background of Network Slices concepts, their lifecycle, how to manage them, and examples of services using network slicing. When detailing the network slice management aspects and the functions needed, there is explicit separation

between Communication Service Management Function (CSMF), the entity responsible for translating the Communication Service into network slice related requirements, the Network Service Management Function (NSMF), the entity responsible for managing the NSI and extract existing network slice subnet requirements, and finally the Network Slice Subnet Management Function (NSSMF), the function responsible for managing the NSSI.

The document then elaborates and details various use cases related to slicing management. Some of the analyzed use cases are the creation, activation, deactivation, modification, termination, and management-related operations for Network Slices and Network Slice Subnets. The document also presents potential solutions for various scenarios, based on a list of requirements for managing those slicing entities, such as network slice management, multiple operator coordination, customer service support, network slice (subnet) fault management, performance management, and lifecycle management.

One of the most relevant aspects to retain from this standard is the solution presented for the multiple operator coordination. It proposes three different approaches to this problem: the first one relies on a communication service deployed across distinct operators; the second approach is a multi-operator slice creation by an operator management system to management system interfaces; the last approach is a multi-operator slice creation by an operator NSMF to NSSMF interfaces.

Network Slice Lifecycle Management Model[7]

The authors in this paper focused on studying the NFV-based Network Slicing Management proposed by 3GPP, identifying a lack of support for virtual Mobile Network Operator (vMNO), proposing an extension over the network slicing management capabilities proposed by 3GPP.

The article mentions that a Mobile Network Operator (MNO) can offer and provide network slices to its customers, which can be vertical industries or virtual Mobile Network Operators. The vMNO, in turn, can offer and provide network slices to its customers, leveraging the infrastructure requested to the MNO. In this scenario, when the vMNO needs to integrate network elements of its infrastructure, it may need to access the MNO NSSMF, causing several problems. For that reason, the authors propose the deployment of the entire 3GPP slice management stack(CSMF, NSMF and NSSMF) as a VNF packages in each vMNO domain. This vMNO slice management stack should maintain the logical association with the MNO central management components.

Finally, this article also proposes an alternative for the VNF scaling problem, suggesting that if a VNF is saturated, the system should search for another VNF providing the same type of service with available resources. If that VNF exists, the system starts balancing the traffic between the two until the saturated VNF returns to its normal state. If that VNF does not exist, the system should trigger the traditional scaling process.

TS 28.541[11]

This 3GPP standard has crucial management and orchestration information, specifically the 5G Network Resource Model. This document presents an extensive specification of the Information Model for 5G New Radio (NR), NG-RAN, 5G Core Network, and network slices.

With a focus on the 5G's radio aspects, the first section of the document details the NR New Radio Model (NRM) information model, presenting both the interactions between Information Object Classes and their definitions. The second section achieves the same, but for the 5GC NRM, detailing both the interactions and nuances of the Information Object Classes necessary for the model. The last section is another detailed description of all the interactions and details of all the Information Object Classes of the network slices NRM.

Although all those models are crucial when implementing a 5G enabled system, for this dissertation proposes, those related to network slices are the most relevant. The document's last section defines the hierarchy between network slicing entities, defining Network Slice as the higher entity, possibly having multiple Service Profiles and multiple Network Slice Subnets. Each Network Slice Subnet is recursive, meaning that it can be composed by other Network Slice Subnets. Additionally, each Network Slice Subnet has multiple Slice Profiles, Managed Functions, and an optional Network Service. Finally, both the Managed Function and the Network Service are composed of one or more Virtual Network Function. Additionally, the information model for each of the referred entities is also detailed, defining the fields and types of values accepted.

Lastly, this technical specification provides annexes with the previously mentioned information models implemented using different data format technologies, such as XML, JSON, and YAML.

NFV-EVE 12 [12]

Published by ETSI as an evaluation report over network slicing support with the ETSI-NFV architecture framework, the document's first section overviews that architecture and the network concepts defined by SDOs(NGMN[14], 3GPP[15], and ONF[16]). From that initial contextualization, the mapping between the NFV concepts and the 3GPP network slicing entities is crucial information to retain. As already presented, according to 3GPP norms, a Communication Service uses Network Slices, which contains Network Slice Subnets that can include other Network Slice Subnets; a Subnet can also be composed by network functions. From the ETSI perspective, a Network Service can contain other Network Services, each one comprised of VNFs or PNFs. The mapping established by this standard is that the 3GPP Network Slices and Network Slice Subnets are equivalent to ETSI Network Services.

The rest of the document extensively analyzes various use cases related to network slicing. Some of those scenarios are the creation of Network Slices and Network Slice Subnets, Network Slice as a Service, and a Network Slice Instance across multiple operators. For each case, the document gives a description, security implications, reliability implications, relation to NFV constructs, and the potential impact on the NFV architectural framework. The last use case is very relevant for the work of this thesis, and as explained in this standard, it derives from a similar scenario presented in TS 28.801[6].

[14] https://www.ngmn.org/
[15] https://www.3gpp.org/
[16] https://opennetworking.org/

Lastly, the document presents recommendations to support network slicing in the ETSI architectural framework, security needs, and reliability aspects.

SOL005 [13]

This ETSI standard is an extensive document detailing protocols, data models, and RESTful Application Programmable Interface (API) protocol specifications for NFV. It specifically defines the RESTful API interfaces for NSD Management, NS Lifecycle Management, NS Performance Management, NS Fault Management, and VNF Package Management provided by the NFVO to the OSS/BSS.

Each of those interfaces is composed of different operations over different resources. Those compositions are:

- **NSD Management**: create, update and query NSD infos, upload, fetch and delete NSDs, create, update and query, Physical Network Function Descriptors (PNFDs) info, upload, fetch, delete PNFDs, create, query and terminate of subscriptions and a notify operation.
- **NS lifecycle Management**: creation and deletion of NS identifiers, instantiate, scale, update, query, terminate and heal NS, create, query and terminate of subscriptions and a notify operation.
- **NS Performance Management**: creation, query and deletion of performance management jobs, creation, query and deletion of thresholds, create, query and terminate of subscriptions and a notify operation.
- **NS Fault Management**: fetch alarms list, acknowledge alarm, create, query and terminate of subscriptions and a notify operation.
- **VNF Package Management**: create, query and update VNF packages info, upload, delete and fetch VNF packages, create, query and terminate of subscriptions and a notify operation.

For each of those interfaces, the document presents the RESTful API operations' hierarchical structure, the flow charts with the interactions between the NFVO and the OSS/BSS, and the detailed data model and HTTP method for each operation.

SOL006[14]

This ETSI standard is a concise document that focuses on protocols and data models for NFV, specifically with descriptors based on YANG specification. Initially, a contextualization and overview of the YANG model, its benefits, drawbacks, and known conventions is presented. Additionally, the document presents and analyzes some rules and concepts related to using YANG models in NFV descriptors.

Finally, the data model definitions for VNFDs, NSDs, PNFDs, Descriptors, and other auxiliary models are detailed. The document also presentes some examples with the proposed models for ease of use.

With the NFV environment constantly growing in complexity, orchestration becomes a crucial aspect to successfully develop 5G networks. For that reason, many researchers from both industry and academic backgrounds are interested in this problem, studying how to solve and improve it. Several works developed in this area focus on the challenges needed to be addressed and available solutions. That is the case for [15], which evidences several challenges in the management and orchestration of 5G services, namely the need for the support of different network hardware/software suppliers efficiently and flexibly, the need for a suitable security system for the 5G trust model, and the need for integrating the new management functionalities with existing OSS/BSS systems. Some of the most relevant orchestration platforms the article presents are OSM and Open Network Automation Platform (ONAP). Those frameworks are open-source projects that are currently in development and actively trying to improve and solve all orchestration challenges.

Another example is the study presented in [16], which compared the most known management and orchestration solutions. The article's authors focused mainly on SONATA[17], a MANO developed within the 5GTANGO 5G PPP project, comparing it with other commonly used MANO platforms, such as OSM and Cloudify[18]. The first comparison study conducted by the authors focuses on the functionalities supported by each orchestrator, targeting areas such as exposed APIs, NS lifecycle management, supported infrastructure abstractions, monitoring capabilities, slice management support, Service Level Agreement (SLA) management, and policy management. All orchestrators were similar in supported functionalities, some applying extra efforts to certain aspects. Finally, the article also presents the performance tests conducted on all three orchestrators. The results were similar between orchestration platforms, proving that all are valid systems to use in NFV-based scenarios.

With the development and definition of Network Slices, the orchestration processes become even more complex. New MANOs are developed with a focus on orchestrating those entities, such as SliMANO[17], 5GTANGO[18], and NESMO[19]. Already existing MANO solutions also make efforts to support networking slicing orchestration, such as OSM, Open Baton, and ONAP.

2.6.1 Available Orchestrators

OSM

OSM[19] is an ETSI-hosted open-source project, aimed at implementing an NFV MANO stack, which involves relevant network operators, cloud operators, and research and academic centers. This community-led project is the go-to platform that complies with the different ETSI defined standards. Although primarily built to be a NFVO, ETSI compliant slice management and orchestration capabilities were added to the platform. OSM currently supports different VIM

[17] https://sonata-nfv.github.io/
[18] https://cloudify.co/
[19] https://osm.etsi.org/

platforms, such as: OpenVIM[20], OpenStack[21], VMware[22], Amazon Web Services (AWS)[23], Microsoft Azure[24], and Eclipse fog05[25].

Open Baton

Open Baton[26] is an open-source platform led by Fraunhofer Fokus[27] and TU Berlin[28] that provides a comprehensive implementation of the ETSI-NFV MANO specification. This platform uses Topology and Orchestration Specification for Cloud Applications (TOSCA) as an alternative template description language. Being initially developed as an NFVO, an external component was created to enable the support of network slice orchestration in this platform. Open Baton currently supports OpenStack, AWS and Docker[29] as VIM platforms.

ONAP

ONAP[30] is an open-source NFVO project backed by the Linux Foundation that provides policy-based orchestration and management capabilities for both physical and virtual networks. This platform's main goal is to provide a common automation platform for telecommunication, cable, and cloud service providers, enabling the automation of different lifecycle processes. This orchestrator also has network slicing management capabilities aligned with 3GPP and ETSI norms. ONAP currently support the following VIM platforms: OpenStack, Kubernetes[31], AWS, Azure, and VMware.

SliMANO

Slice MANO(SliMANO)[17] is an ETSI-complaint plug-in based system, able to manage and orchestrate E2E network slices. Compared to the already mentioned orchestrators, this platform is able to connect to various network orchestration entities, such as NFVOs, SDN, and RAN controllers. Being plug-in-based, SliMANO can be platform agnostic and guarantee the interoperability with different controllers and NFVOs.

5GTANGO

The 5GTANGO[18] project proposes a NFVO framework architecture aimed at bringing MANO to multiple VIM domains, allowing the orchestration of appliances on multiple VIMs interconnected using WIM. This orchestrator was developed with a strong focus on policies, SLAs, and their management. In addition to the NFVO capabilities, this platform also has network slicing management capabilities based on the internal NFVO framework. For that

[20]https://www.openvim.com/
[21]https://www.openstack.org/
[22]https://www.vmware.com
[23]https://aws.amazon.com/
[24]https://azure.microsoft.com
[25]https://fog05.io/
[26]https://openbaton.github.io
[27]https://www.fokus.fraunhofer.de/
[28]https://www.av.tu-berlin.de
[29]https://www.docker.com/
[30]https://www.onap.org/
[31]https://kubernetes.io/

reason, those capabilities are restricted to that NFVO and cannot deploy slices in orchestrators outside the framework's domain.

NESMO

The NESMO[19] project proposes a network slicing management and orchestration framework. This project defines the slice management framework from the network operator's point of view, taking into account the development of mechanisms to automate the design, deployment, and management of network slices. Unfortunately, it does not take into consideration the 3GPP slice management specification [6], and it does not have a proof-of-concept to consolidate its viability.

2.7 Summary

This chapter describes the technologies used to enable 5G networks and systems, and how they are used to fulfill the 5G promises. How technologies like NFV, SDN and Network Slicing bring agility to the deployment of new network services while bringing the costs down.

This chapter also presents the most relevant studies defining network entities, such as VNFs, NSs and NSIs, the possible actions for each one, and how to manage and orchestrate them. The standards, in particular, are crucial for this thesis since they are the basis for the models and operations supported by the POC developed.

All 3GPP technical specifications helped understand the network slicing concepts, hierarchies, operations, and management considerations. In particular, based on TS 28.801[6], I concluded that the POC developed for this thesis should take the role of the CSMF, communicating with the underlying network slicing enabled NFVOs that will act as NSMFs and NSSMFs. Based on TS 28.541[9], I gathered which information models the POC should support concerning network slices.

The ETSI standards helped to better understand the NFVO side of things. The NFV-EVE 12[12] helped me understand how the 3GPP slicing concepts are mapped to the ETSI-NFV architecture. After grasping that knowledge, by analyzing the SOL005[13] and SOL006[14] standards, I was able to understand which information models and operations the POC should support concerning the NFVO level.

Finally, the problem of how these new entities are managed and orchestrated is also addressed, presenting the current orchestration challenges and available solutions. By using a MANO system, it simplifies the entire process.

CHAPTER 3

Vertical Service Orchestrators

With the evolution of the 5G technologies, the improvements in network slicing and its orchestration, and the continuous standardization by SDOs, many projects were created to provide orchestration solutions for vertical industries.

By having vertical industries as its users, these projects needed to simplify the interaction between the verticals and the orchestration system, possibly abstracting the network complexities. Another important aspect these projects needed to consider was the restrictions verticals services have, meaning that all orchestration systems should have a way of defining and maintaining a certain QoS. Finally, since interacting with industry players, it's paramount to follow the correct norms, hence the need for new and improved standards.

From my knowledge, the most relevant projects with vertical service orchestrators are OpenSlicer, 5G-Transformer, and 5Growth. All three projects will be detailed in the following sections.

3.1 OpenSlicer

OpenSlicer is a project that originated from the developments and achievements of another project, the 5GinFIRE [20].

To give a brief contextualization about that project, 5GinFIRE is a project funded by the European Horizon 2020 Programme for research, technological development, and demonstration under grant agreement n° 732497. According to [21], its objective was to establish and manage an Open 5G NFV-based ecosystem to create experimental architectures and facilities, enabling testing solutions for vertical industries. Additionally, the project set out to perform some integrations with other Horizon 2020 projects, such as Future Internet Research and Experimentation (FIRE) and 5G-PPP projects. FIRE is an initiative that "facilitates strategic research and development of new Internet concepts, giving researchers the tools they need to conduct large-scale experiments on new paradigms" [22].

For that purpose, a system was designed and developed, resulting in the architecture present in Figure 3.1. The main components of the said architecture are:

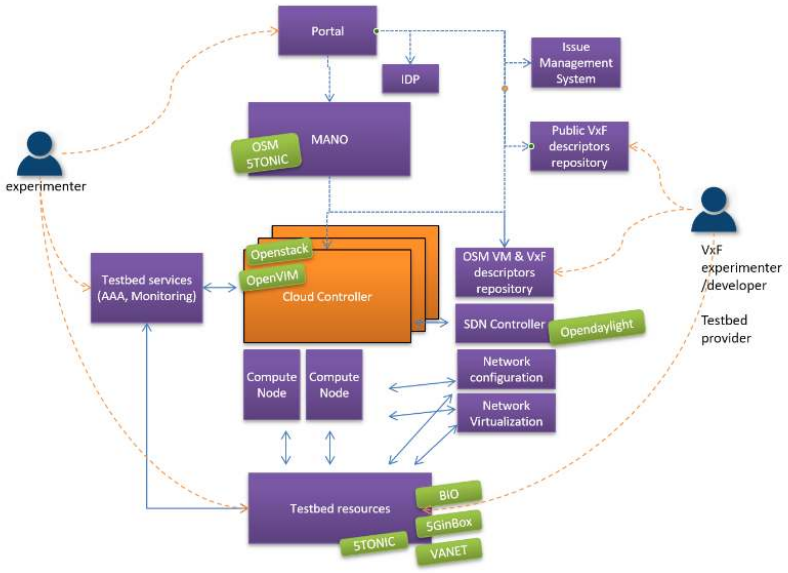


Figure 3.1: 5GinFIRE high level architecture [20]

- **Portal**: Based on a web application, both experimenters and developers can interact with the system to browse repositories or subscribe, manage and monitor experiments. Admins have special access to the platform and repository management services. Also, it has Authentication, Authorization and Accounting (AAA) mechanisms that allow integration with other FIRE facilities and the seamless acceptance of their users, allowing the creation of federated experiments.
- **IDP**: The Identity Provider (IdP) is responsible for storing and providing identifiers and information about the platform users the rest of the system.
- **Public VxF descriptors repository**: As the name implies, this component hosts a catalogue of all registered Virtual Functions (VxF) descriptors, allowing continuous availability.
- **5GinFIRE MANO platform**: This component is one of the most important on the NFV system, as defined by ETSI reference architectural framework. Based on OSM, this component allows the creation and deletion of VNFDs and NSDs, and the instantiation of NSs.
- **Issue Management System**: Based on Bugzilla, this Issue Management System allows tracking issues and necessary notifications, integrating with the Portal to enable the automatic creation and update of issues.
- **VIMs (Cloud Controllers)**: Each partner hosting a given experimental facility must deploy and manage the required VIM. Although controlled by the partner, the VIM must be one of two supported by 5GinFIRE: OpenVIM or OpenStack.
- **Testbed Services**: According to [21], these services "are some testbed-specific services that could be handover to the experimenter to ease the operations during experimentation".
- **Testbed Resources**: According to [21], these resources relate to "the available resources for experimentation located in each target testbed".

With the developments achieved, the OpenSlicer originated as 5GinFIRE spinoff, taking advantage of the most relevant components that constitute an open-source OSS/BSS platform, the goal of OpenSlicer. This new project also supports the onboarding of VNFDs and NSDs, the instantiation of NSs, and even has TMForum[1] Open APIs support regarding Service Catalog Management, Ordering, Resource, and more.

The OpenSlicer system defines some abstractions over the VNFs and NSs complexities, allowing verticals to request a given service in the most seamless way possible, focusing only on the service logic. The first abstraction is the Service concept, which consists of the abstraction of all the underlying network and infrastructure complexities, enabling OpenSlicer to offer coherent and correct services, ready to be instantiated and deployed. Other concepts are the Resource Facing Services (RFSs) that encapsulate all services directly connected to the infrastructure (such as the NSDs) and Customer Facing Services (CFSs), which encapsulate all services and specifications that directly interact with the user. Both RFSs and CFSs are fundamental components of the Service concept. To create a Service, a CFS Specification needs to be defined, possibly having Service Specification Relationships attached to connect the RFSs to CFSs. For example, a user "can create a CFS spec called 'A 5G Service' as a bundle of two other services (include them in Service Specification Relationships) such as a 5G eMBB Slice and a Customer VPN. So when the user orders 'A 5G Service' services from 5G eMBB Slice and a Customer VPN will be created during the order" [23].

OpenSlicer contains two different web portals: the first allows verticals to query/instantiate services and providers to design them; the second web portal allows NFV developers to manage and onboard related artifacts. Lastly, OpenSlicer also has some federation capabilities through the TMForum Open APIs, which enable the definition of Eastbound/Westbound Interface (E/WBI).

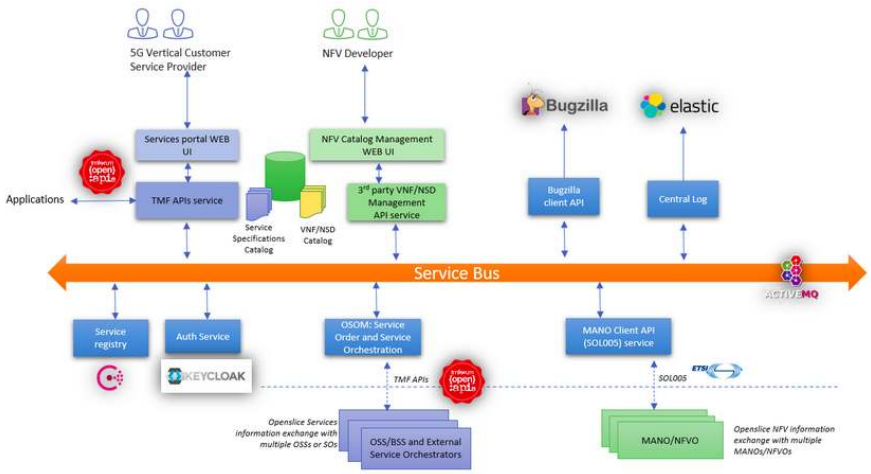


Figure 3.2: OpenSlice high level architecture [23]

The final overall OpenSlicer architecture is the one presented in Figure 3.2, and the main components present in it are [23]:

[1] https://www.tmforum.org/

- **Service Portal Web UI**: The Service Portal is responsible for all operations related to Services. Verticals can browse, query, and instantiate existing services, and service providers can create new service specifications.
- **NFV Catalog Management Web UI**: The Catalog Portal is the one responsible for allowing NFV developers to manage and onboard VNFDs and NSDs to corresponding NFVO facilities.
- **TMF APIs Service**: This component is responsible for providing TMForum compatible services, which allows the exposure of catalogs, acceptance of service orders, and the implementation of E/WBI between the domains.
- **VNF/NSD Management API Service**: This component receives and forwards to the respective entities the requests related to VNFD and NSD onboarding.
- **Service Order and Service Orchestration (OSOM)**: This component is an OpenSlicer custom solution that handles Service-related requests. It is responsible for orchestrating and ordering services, communicating when needed with underlying Service Orchestrators and NFVOs.
- **MANO Client API Service**: This component serves as a client for the supported MANOs and NFVOs, receiving and redirecting NFV related requests.
- **Catalogs**: Like other systems of this kind, some important repositories and catalogs store and manage many of the system's assets. The Service Specification Repository and the VNFDs/NSDs Catalog are the two main ones. The first repository stores all Service Specifications created. The second catalog stores all the VNFDs and NSDs onboarded.
- **Logging**: This component is responsible for logging the entire system's activities and providing the necessary reports, aiding the platform's decision-making. This component consists in a central micro-service based on an Elasticsearch[2] cluster.
- **Issue Tracking Client**: This component provides an interface through which an issue can be registered in the issue management service, enabling the posterior notification of said issues to the interested parties.

As advantages, this platform has the high-level Service abstraction, the separation of customer/resource facing services, and the specialized distinct Web Portals. On the other hand, the lack of network slicing capabilities, the minimal multi-domain support, and the monolithic architecture are its disadvantages.

3.2 5G-Transformer

The 5G-Transformer project is a 5G-PPP phase 2 project created to improve the current mobile transportation network by transforming it into a SDN/NFV-based Mobile Transport and Computing Platform (MTP), integrating network slicing on mobile transport networks [24]. Furthermore, this integration allows the creation and management of MTP slices, specially tailored for a given vertical industry. According to [24], the following two points summarize the objectives of the project:

[2]https://www.elastic.co/

1. Enable vertical industries to meet their service requirements within customized MTP slices; and
2. Aggregate and federate transport networking and computing fabric, from the edge all the way to the core and cloud, to create and manage MTP slices throughout a federated virtualized infrastructure.

Given those objectives, there was the need to design and implement a 5G platform capable of demonstrating the previously mentioned goals. As a result, the final system follows the architecture presented in Figure 3.3, containing three main components: the Vertical Slicer, the Service Orchestrator, and the Mobile Transport and Computing Platform.

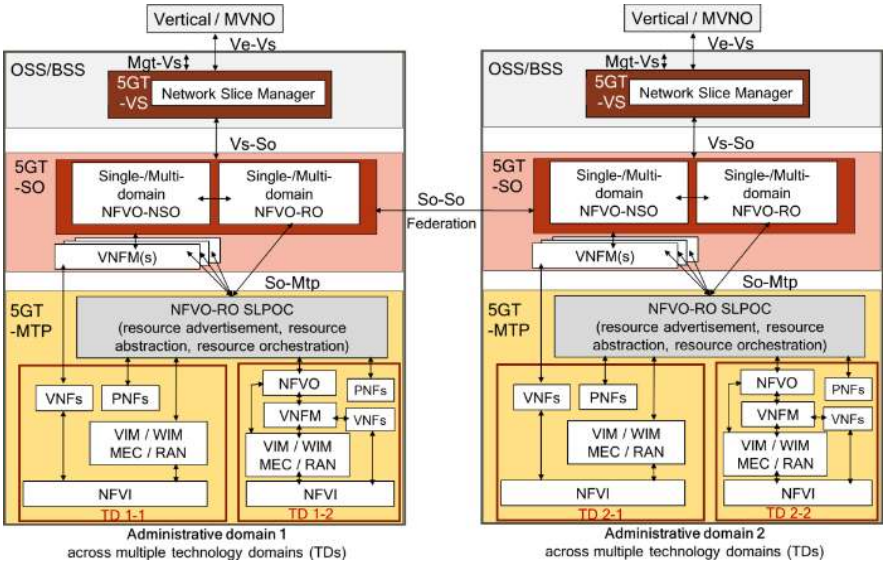


Figure 3.3: 5G-Transformer high level architecture [25]

3.2.1 VS

The 5G-Transformer (5GT)-Vertical Slicer (VS) is the entry point for verticals into the system and an OSS/BSS component of the 5GT administrative domain. This entity is responsible for coordinating and arbitrating vertical services, accessible to verticals through a high-level interface focused on their logic and needs [26]. A vertical must establish a SLA with the 5GT platform, specifying a series of Service Level Objectivess (SLOs) that will indicate the intended QoS values for their services(for example, the maximum E2E latency of 20 ms). Any SLA degradation, such as a decrease in their services performance may cause severe problems to verticals. In some cases, such degradations can impact the reputation and business leadership of the vertical.

For the high-level abstraction needed by verticals, the 5GT platform uses VSB to define a vertical network service composition, providing a skeleton ready to use when needing that topology. As a VSB extension, a vertical must create a VSD by provided the QoS parameters for is scenario, resulting in a ready-to-use deployment recipe that will instantiate the defined topology and follow the given QoS values. The platform maps each VSI onto network slices,

which in the 5GT system are an extension of the ETSI NFV NSDs [26]. Different VSIs can share the same network slices.

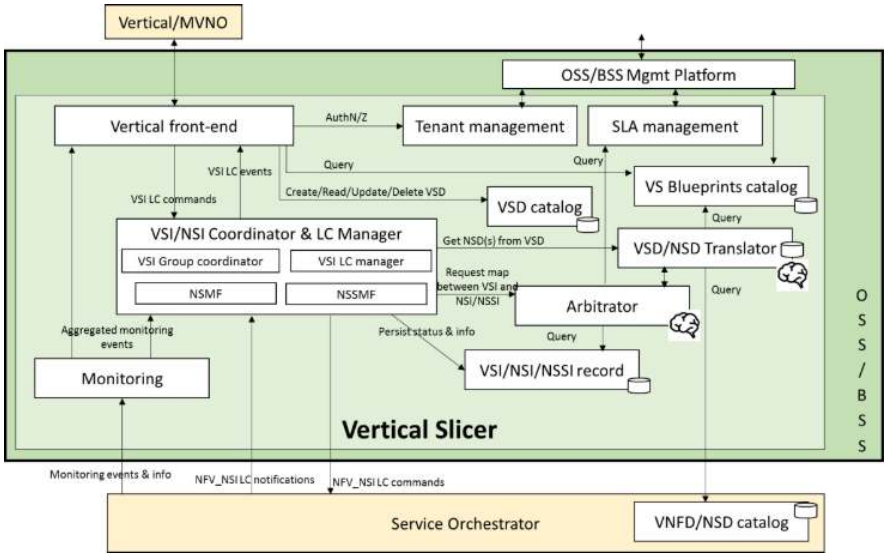


Figure 3.4: 5G-Transformer VS high level architecture [26]

Being on top of the Transformer stack, the 5GT-VS exposes both an NBI and an SBI. The NBI is used to interact with verticals and Mobile Virtual Network Providers (MVNOs), and the SBI to interact with underlying Service Orchestrators (SOs). As presented in Figure 3.4, 5GT-VS has many relevant components in its architecture, such as:

- **VSD/NSD Translator Module**: This component is responsible for mapping the VSDs into NSDs compatible with the 5GT-SO underneath. Some of the actions required to achieve that mapping are validating that the VNFD catalog contains the referenced VNFs, validating that the NSD catalog includes the referenced NSs, and completing the deployment flavors and VLDs configurations considering the VSD's QoS values.
- **Arbitrator**: This component is responsible for ensuring the SLA defined between the vertical and the 5GT platform is respected. To achieve this, the Arbitrator has two main tasks, deciding how to map the VSIs in NSIs/NSSIs and determining the deployment flavors for each service according to QoS requirements.
- **Monitoring**: This component is responsible for generating monitoring data about the network slices, vertical services, VNFs, and NSs. The underlying 5GT-SO provides the information for the last two entities. The gathered information is exposed to the verticals through the 5GT-VS NBI and can also be used internally to aid the Arbitrator's decision-making process related to resource arbitration. The Monitoring component can also supervise alarms related to NSIs and VSIs, as long as it is subscribed to them. Finally, it enables the definition of thresholds concerning a specific performance metric, notifying the interested entities when triggered.
- **VSI/NSI Coordinator & LC Manager**: This component is responsible for coordinating the entities generated by the 5GT-VS, such as VSIs, NSIs, and NSSIs. There are three different management domains, the coordination of the VSIs of a given vertical,

the coordination of a single VSI, and the management of both NSIs and NSSIs. The VSI Group Coordinator handles the first domain, which considers the limitations defined for the vertical's VSIs (maximum CPUs, memory, and storage available) and ensures those restrictions are respected. The VSI LifeCycle Manager (LCM) handles the second domain, which keeps track of the requested VSIs and the associated NSIs supporting them, keeping track of the resource consumption for future accounting. Both the NSMF and the NSSMF handle the third domain, respectively managing NSIs and NSSIs. Those network management functions enable the deployment of VSIs and the assessment of their feasibility, controlling the associated NSIs and NSSIs based on the PNFs and VNFs described in the NSDs. They also keep track of the resources required per NSI, information used by the Arbitrator.

Furthermore, a second design phase defined and selected additional refinements and extensions for this component. Some extensions are: direct feedback from vertical applications to enable the communication between vertical services and 5GT-VS; service configuration allowing updating application-level parameters per instance; vertical-driven service composition allowing verticals to compose complex services from basic building blocks; and better policy management aiding SLA enforcement. This second design phase also defined extensions to improve autoscaling by adding scaling rules to the NSDs, to allow the creation of composite services by the providers, and to improve the advanced management of network slices composed of multiple slice subnets.

The advantages of the 5GT-VS are the VSBs and VSDs abstractions over the underlying complexities, the initial support of network slicing with extended NSDs, the multi-domain support, and the robust SLA management. As for disadvantages, the network slicing support is not according to current standards, the multi-domain support is only between Service Orchestrators, and the architecture is monolithic.

3.2.2 SO

Located in the middle of the 5GT stack, the 5GT-SO communicates with 5GT-VS through its NBI and through its SBI for the 5GT-MTP. This component is also able to communicate with other SOs through its E/WBI. This component is responsible for orchestrating the E2E deployment of NSs across one or more administrative domains, addressing and managing the allocation of different VSIs [26]. For single-domain local instantiations, the 5GT-SO communicates with the local 5GT-MTP. For inter-domain, the MTP interacts with 5GT-SOs from other domains, enabling the inter-domain feature.

Given the NSDs provided by the 5GT-VS, the 5GT-SO assigns the necessary virtual networking, computing, and storage resources across one or more 5GT-MTPs to fulfill the specified service requirements. The Network Service Orchestrator and the Resource Orchestrator are two relevant SO components used for single and inter-domain scenarios, which according to [26], are functionally equivalent to the ETSI defined NFVO. The Network Service Orchestrator handles the deployment coordination of NSs and their lifecycle. The Resource Orchestrator handles the orchestration of virtual resources across one or more domains.

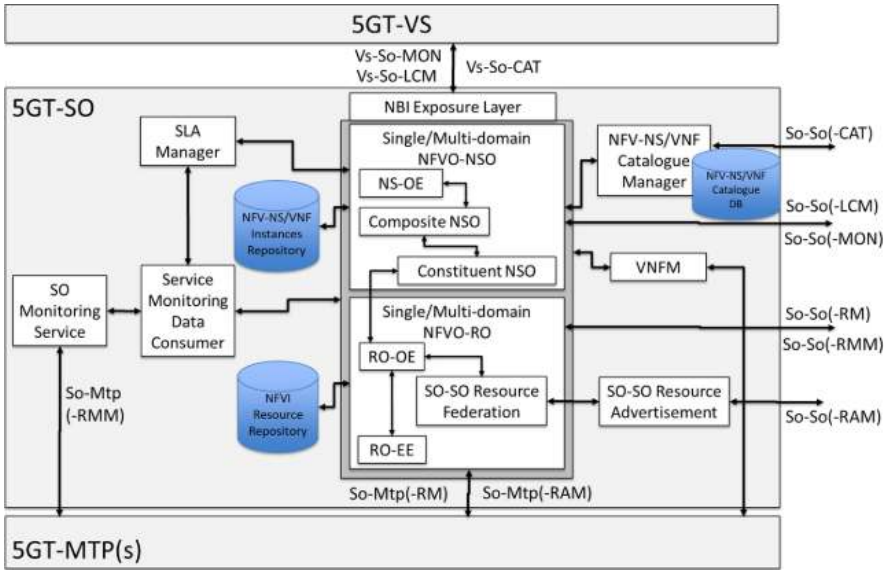


Figure 3.5: 5G-Transformer SO high level architecture [26]

As seen in Figure 3.5, several components comprise the 5GT-SO, where the most important ones are:

- **NFV Orchestrator (NFVO)**: This component is composed of the already mentioned Network Service Orchestrator and Resource Orchestrator. The Network Service Orchestrator(NSO) coordinates the deployment and lifecycle of NSs, segmenting the NSDs into Composite NSOs according to the Network Orchestration Engine algorithms. This functionality facilitates the inter-domain deployment. On the other hand, when the Resource Orchestrator receives NSD segments, it maps each one onto a set of virtualized infrastructure resources by deciding the placement of each VNF.
- **VNF Manager (VNFM)**: This component is responsible for managing the lifecycle of the 5GT-SO deployed VNFs (that use local, remote, or both types of resources). Besides, it can also receive lifecycle events and provide reconfiguration actions according to previously defined counter-actions.
- **Monitoring**: The monitoring system contains two 5GT-SO entities: the Monitoring Service and the Monitoring Data Consumer. The Monitoring Service provides measurement reports that the SO uses to "adapt deployed services or provisioned resources while preventing service degradations and/or SLA violations" [26]. The Monitoring Data Consumer collects said measurement reports and forwards the data to the NFVO, which can trigger auto-scaling actions, healing actions, etc.
- **SLA Manager**: This entity is responsible for creating performance reports from the Monitoring Data Consumer and enforcing agreed SLAs. If there is an SLA violation, this entity can trigger scaling actions to correct the mentioned problems.
- **Repositories & Catalogues**: There are several repositories and catalogs present in the 5GT-SO architecture, such as the NFVI Resource Repository, the NS/VNF Instance Repository, and the NFV-NS/VNF Catalogue DB/Manager, each one responsible for storing and managing a given set of entities. The NFVI Resource Repository stores

consolidated abstracted resource views received from 5GT-MTPs or other 5GT-SOs. The NS/VNF Instance Repository stores the active VNFs and NS instances. Lastly, the NFV-NS/VNF Catalogue DB/Manager stores and manages the NSDs, VNFDs, and Application Descriptors.

Furthermore, a second design phase defined and selected additional refinements and extensions for this component. A core extension was the improved service scaling by offering service scaling at runtime without service interruption, which could be triggered by the vertical at the 5GT-VS, performed automatically by the VS's Arbitrator or performed by the 5GT-SO according to NSDs rules. Some other extensions were the updated architecture to support the new scaling options, which demanded extensions on the Monitoring Manager and an SLA Manager. Concerning service composition and federation, the second design phase also considered extensions over the Constituent Network Service Orchestrator for nested NSs, the Composite Network Service Orchestrator to decompose NSDs, and the Link Selection Algorithm to improve the network links computation in federation scenarios. Finally, the second design phase also defined Placement Algorithms to improve inter-domain service orchestration and location restraints.

3.2.3 MTP

Being at the bottom of the 5GT stack and only communicating with 5GT-SO through its NBI, the 5GT-MTP is the foundation of the system. This component is responsible for hosting and managing both physical and virtual network, computing, and storage resources, allowing the deployment of vertical services.

This entity uses both SDN and NFV capabilities to simultaneously support a diverse range of networking and computing requirements, allowing a personalized fit for vertical industries. The 5GT-MTP system provides two main functionalities: the first one is the coordination and provisioning of radio, transport, storage, and computational resources required by the different vertical services; the second is the support of an abstracted and unified view of the supported resources, hiding the complexities and the diversity of technologies needed. This component supports various VIMs and WIMs, even if implemented with different technologies, always exposing a unified view for the upper layer. Those infrastructure managers can, in turn, communicate with the underlying infrastructure when needed, returning to the 5GT-MTP that manages all information.

According to [26], 5GPPP Phase 1 projects, specifically the 5G-Crosshaul[3], and ETSI NFV standards heavily influenced the design phase of this component. Its main objective was to successfully support the upper layers' logic and allow efficient infrastructure resource usage. Because of that, the 5GT-MTP NBI, which enables the communication with the 5GT-SO, follows standards such as ETSI GS NFV-IFA 005, ETSI GS NFV-IFA 006, and ETSI GS NFV-IFA 008. Concerning the internal SBI, that enables the communication between 5GT-MTP and the underlying VIMs, it follows standards such as ETSI GS NFV-IFA 006 and ETSI GS NFV-IFA 013.

[3] https://5g-ppp.eu/xhaul/

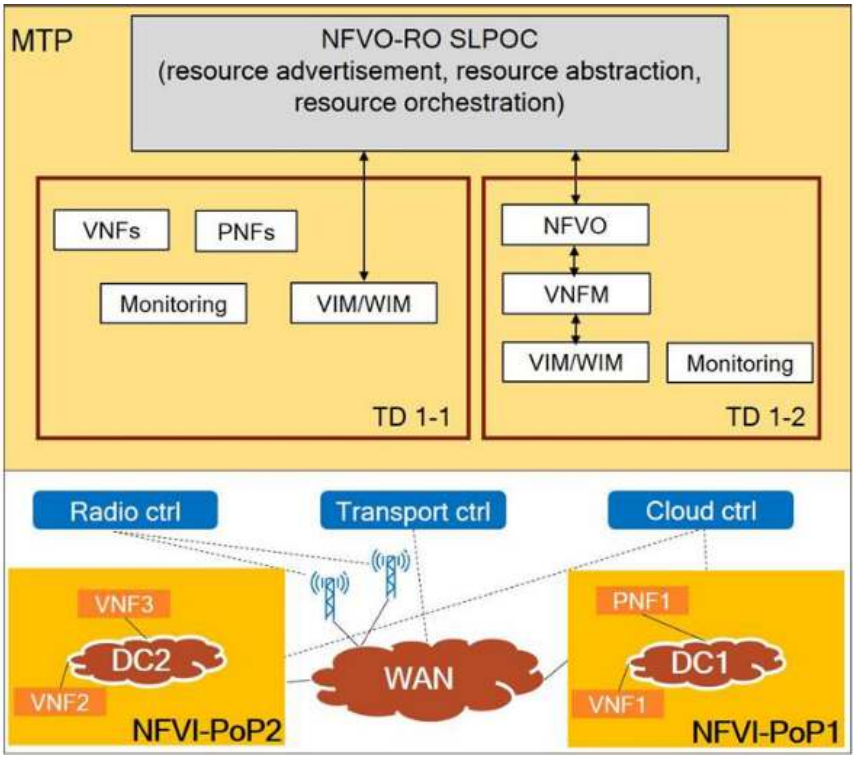


Figure 3.6: 5G-Transformer MTP high level architecture [26]

As shown by Figure 3.6, the main components of the 5GT-MTP architecture are:

- **Single Logical Point of Contact (NFVO-RO SLPOC)**: This component serves as a single point of access for the upper layers and provides a suitable, abstract, and unified view of all resources. Since this component receives the resource allocation requests, it is also responsible for orchestrating the resources and ensuring the selection and configuration of transport, radio, compute, and storage resources are correct. This entity needs to communicate with the respective management entities to allocate the resources, such as VIMs.
- **VIMs**: These components are responsible for managing storage, computational and networking resources in a respective NFVI-PoP. The VIM is typically handled by a cloud platform, many times containing one or more SDN controllers to establish transport connectivity between the different VNFs deployed in a given PoP.
- **WIMs**: These entities are responsible for providing inter-domain links, which according to [26], consists of "configurations of the transport network between NFVI-PoPs gateways through the proper SDN controller".
- **NFVIs**: These entities are responsible for providing all the hardware, such as storage, compute, and networking resources. Furthermore, they must also support the needed software, such as the hypervisors used to create the infrastructure for future VNFs.
- **Monitoring**: This component is responsible for gathering, monitoring, and providing all the data generated from the different domains concerning radio, transport, cloud, physical, and virtual resources.

Furthermore, a second design phase defined and selected additional refinements and extensions for this component, mainly focused on RAN and Multi-access Edge Computing (MEC) support. Those extensions were the update of 5GT-MTP interfaces and information models to support RAN and MEC configurations, the addition of slice support for transport networks with heterogeneous technologies, the extension of the MTP monitoring system, and the introduction of local Placement Algorithms to handle resource selection inside a given technology domain.

3.3 5Growth

5Growth is a 5G-PPP phase 3 part 3 project, one of the eight composing this phase part, that tries to push the 5G vision of "5G empowered vertical industries" closer to deployment. 5Growth defined that its objective was to validate the 5G technologies from the verticals point of view, both technically and business-wise. For that reason, the project decided to take advantage of the achievements and developments in network slicing, virtualization, and multi-domain solutions of the 5G-PPP phase 2 projects, such as 5G-Transformer and 5G-Monarch [4]. Furthermore, the project chose two ICT-17-2018[5] 5G E2E platforms to test their developments, namely 5G-EVE[6] and 5G-VINNI[7] [27]. 5Growth set out to support industry vertical processes by providing four main features: a vertical portal for bridging the gap between verticals and the 5G facilities; closed-loop automation; SLA control for the services' lifecycle; and finally, an Artificial Intelligence (AI)-driven E2E network solution to "optimize Access, Transport, Core and Cloud, Edge and Fog resources, across multiple technologies and domains" [27].

The 5Growth (5GR) project used the 5G-Transformer platform as the starting point where extensions/enhancements over its composing blocks were added (5GT-VS, 5GT-SO and 5GT-MTP). The improvements follow both functional and service requirements of the use cases devised for the project. The platform work plan consisted of different innovations, each selected to fill a given gap found in the base platform, resulting in twelve improvements.The design and intended 5Growth architecture is the one present in Figure 3.7, and the innovations that led to that architecture are [28]:

1. **Support of Radio Access in network slices**: Network Slices typically span across CN and AN, being RANs used in most cases. Since the 5GT platform only supports NSs and their composing VNFs and PNFs, entities responsible for the CN segment, that system does not support RAN. The 5Growth platform needed to extend the information models, such as VSBs, VSDs, and NSTs to support RAN reservation and configuration. That extension consisted in adding new fields related to the service slice category(ex. eMBB, URLLC, mMTC) and RAN parameters. Besides, if the 5GR-SO also manages RAN-related topics, models such as the NSDs will also need modifications.

[4] https://5g-monarch.eu/
[5] https://cordis.europa.eu/programme/id/H2020_ICT-17-2018
[6] https://www.5g-eve.eu/
[7] https://www.5g-vinni.eu/

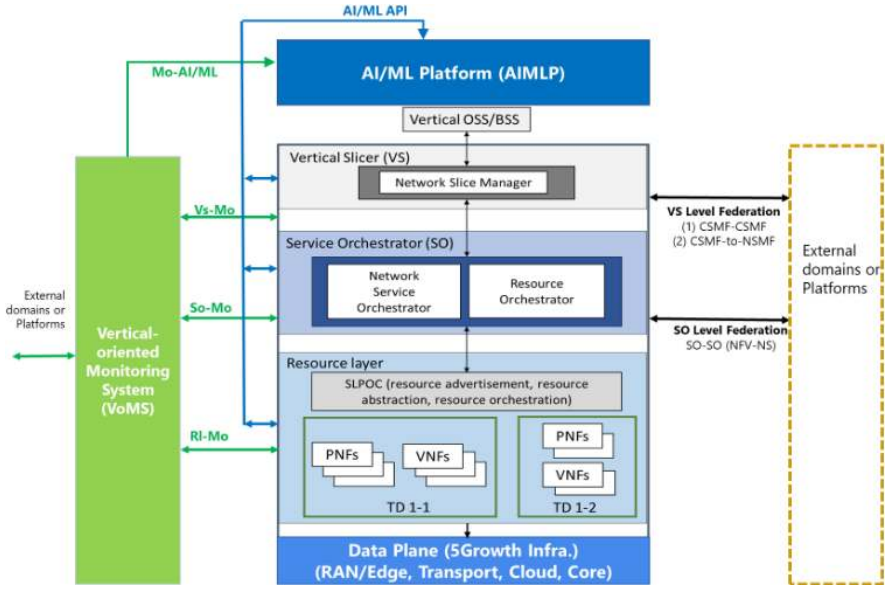


Figure 3.7: 5Growth high level architecture [29]

The architecture itself also needs to be extended to support reservation and configuration of RAN resources, namely enhancing interfaces and internal procedures. Finally, the 5GR-Resource Layer (RL), an extension of 5GT-MTP, may establish different levels of RAN abstractions.

2. **Vertical-oriented Monitoring System**: As part of the inherited Monitoring Platform developed in 5GT, this new entity will add advanced metric collection capabilities by using dynamic probes and AI-based control over the 5GR's vertical services. This innovation selected Message Queues to enable the exchange of messages between the monitoring agents, present in the vertical services, and the the 5GR monitoring service. Processing and analyzing the collected data can improve scaling and self-healing operations. Data processing modules (ex. AI/Machine Learning (ML) platform or forecasting module) can also use the generated monitoring data to build models and improve the system's decision-making.
3. **Monitoring Orchestration**: The Monitoring Orchestrator is a 5GR-SO module that manages different monitoring functions needed for heterogeneous services/slices, guaranteeing performance and SLA requirements. This orchestrator also allows the dynamic deployment of monitoring probes, auto-discovering probing possibilities by analyzing the collected monitoring metadata. Entities such as the 5GR-SO or the 5GR-VS can subscribe to this service, receiving parameters and notifications obtained by the Monitoring Orchestrator based on thresholds. Finally, this entity also manages the lifecycle of monitoring function instances, like Message Queues, Time Series Databases, visualization services, and more.
4. **Control-loops stability**: The goal with this innovation is, as the name suggests, to provide "closed-loop automation and SLA control for vertical services lifecycle management throughout the 5GR system" [28]. Closed-loop in this context means that

the data produced by the 5GR system is reintroduced in the platform to improve and correct any problems detected. The steps in this process are: in the first place, collect monitoring data from services and networks, followed by processing it with real-time data analytic tools to infer events and alarms, as well as provide recommended actions with the help of AI/ML; the last step is to make the correct decisions for optimizing and reconfiguring the services, achieved by triggering auto-scaling, self-healing or traffic management actions. This entire process is cyclical, meaning that after the third step the process should start again.

5. **AI/ML Support**: AI/ML algorithms proved to be ideal to deal with many of the problems faced by the 5GR system, specifically in the SLA and performance requirements enforcement. Those algorithms allow the 5GR platform to adapt to the dynamic behavior of hosted heterogeneous services/slices, enabling a more insightful decision-making process through the real-time information given by the AI/ML algorithms. The objective of this innovation was to support and aid AI/ML algorithms, both those developed for this or other innovations. The first goal was to identify a suitable location in the 5GR system to insert a new module, responsible for hosting the various algorithms that interact with the monitoring and orchestration components. Another goal of this innovation was to identify proper training and testing datasets with the aid of verticals. Finally, there was also the need to define the best architectural approach to support the different AI/ML techniques.
6. **Federation and inter-domain**: Federation establishes inter-domain with a previous knowledge concerning which domain will be connected, differing from full-fledge inter-domain that does not consider any prior knowledge. With those concepts, the 5GR project will extend its capabilities and offerings by aggregating both service and resource catalogs from different peering providers, each with its NFVO. The services that can take advantage of these concepts are communication services, network slices, and NFV network services. Each service type may need a different approach, resulting in the inter-domain and federation support spanning across different abstraction layers. For instance, the 5GR-VS should handle the communication services and network slices, using the CSMF and NSMF capabilities defined by 3GPP. Peering 5GR-SOs are responsible for the NFV network services and resource federation.
7. **Next-Generation Radio Access Network**: This innovation was selected to support the Next-Generation RAN and focus on virtual RAN (vRAN). The 5GR project decided to base its vRAN orchestration capacities work in the O-RAN's [8] reference architecture. This organization founded by operators, define requirements and build a supply-chain ecosystem that respects two principles: Openness, which defends that open interfaces are a must for building agile and cost-effective next-generation RANs, simplifying the work for small vendors and operators when adding new services and enabling multi-vendor deployments; the second principle is Intelligence since that 5G's achievements result in very complex networks, the only solution to control such complexity is to enable

[8] https://www.o-ran.org/

self-driven networks that can leverage new technologies, automate operational network actions and decrease its costs.

8. **Smart Orchestration and Resource Control**: 5GR set out to become a platform with fully automated network slice lifecycle management and SLA enforcement, enabled by the architectural innovations, novel algorithms and techniques. This innovation defined two targets in relations to the algorithms developed and used in 5GR: address the dynamic E2E problem by introducing SLA-aware service orchestration/arbitration, adaptive resource allocation, and adding scaling approaches across all three layers of the software stack, aided by AI/ML solutions; the second target was to incorporate in the 5GR-RL dynamic resource control algorithms/techniques for performance management, such as (re)programmable SLA-aware traffic management algorithms at the data plane. The plan was to deploy new algorithms in each entity of the 5GR stack: new service arbitration algorithms in the 5GR-VS; new single and inter-domain service and resource orchestration algorithms in the 5GR-SO; new dynamic resource allocation and re-optimization algorithms, allow data plane customization, enable performance isolation per slice and enable programmable data plane arbitration in the 5GR-RL.
9. **Anomaly Detection**: Due to the heterogeneous nature of the services supported by the 5GR platform, admins and tenants need an anomaly detection system to help them detect and diagnose problems. This innovation proposed the creation of a flexible AI-driven module to analyze network information provided by the Monitoring Platform, identifying possible anomalies and their root cause. By analyzing historical data, this component can find and add new exceptions to its internal anomaly set. It can also improve the system's recovery time of known problems. Some features introduced by this module are new algorithms for real-time anomaly detection in the 5GR-SO, new algorithms for root cause analysis, and new prediction algorithms based on identified anomalies and root causes.
10. **Forecasting and Inference**: A prediction and forecasting system proved to be a component needed on the 5GR platform to ensure the strict verticals' use cases requirements. Inferring allows pre-emptive adjustments of the offered services through auto-scaling, self-healing, or self-reconfiguring services, mitigating anomalies impact and preventing service downtime. This innovation aimed at developing efficient and effective algorithms that allow the forecasting and inference of events based on data analytics. The aim was to achieve the "prediction of resource usage to avoid congestion; prediction of service KPIs to optimize service; prediction of service health state to ensure reliability; etc." [28].
11. **Security and Auditability**: With such a complex system, it should not lack in the security and auditability aspects. This innovation was responsible for proposing efforts in this area, providing a verifiable and trustable stack by protecting the multiple request-response exchanges, both within the vertical environment and across different actors from distinct administrative domains. This innovation proposed the integration of non-repudiation mechanisms and advanced security methodologies into the 5GR

platform. The non-repudiation mechanism allows demonstrating that a user is the creator of a request/response exchange between two entities, allowing its correlation to other messages in a consistent timeline. By securely storing the information of those exchanges, those mechanisms provide the traceability and auditability required. In terms of the advanced security methodologies, the innovation selected Moving Target Defense (MTD) and Cyber Mimic Defense (CMD). The core concept of MTD is establishing a time limit for the validity of the service interfaces by continuously moving and mutating them. By simply changing the IP address or Layer 4 port of the service interface, this mutation process significantly slows a possible attack, helping the detection of attackers present in the network and possibly countering Denial of Service attacks. It can also serve as a honeypot tool, a service with false but convincing information that traps the attacker in a secure location. Finally, the CMD method relies on multiple implementations of the same function by different parties. The service to protect must follow the Input-Process-Output model, meaning that it must be fully deterministic and always return the same output for the same input.

12. **5Growth CI/CD and containerization**: This innovation proposed to improve the 5GR platform lifecycle(developing, testing, integration, and deploying) management automation, allowing the decrease of its time-to-market. Based on Docker [9] and using Kubernetes [10] orchestration, this innovation brings flexibility, reliability, convenience, and management ease that otherwise was not possible. With the help of some open-source, community-approved and production-ready tools, it allowed the implementation of Dev-Ops paradigms such as E2E automation, Continuous Integration (CI)/Continuous Delivery (CD), infrastructure as code, orchestration, and test automation.

Since the 5GR-VS is an extension of the 5GT-VS components, it inherits many of its advantages and disadvantages. In addition to VSBs and VSDs abstractions over the underlying complexities and the SLA management, the 5GR-VS system adds network slicing according to standards and improves the multi-domain support. On the other hand, the architecture remains monolithic, and since the implemented innovations were add-ons, the final platform's quality can be lower than expected.

3.4 Summary

In this chapter, the most relevant service orchestration platforms were presented and analyzed. For each project, the advantages and disadvantages of the orchestration component were extracted, concluding that there are positive and negative aspects spanning across more than one project. These observations helped in some decisions for the development of this thesis' POC.

Concerning the positive aspects gathered from all projects, the aspect that stood out was the network and infrastructure abstractions adopted by all platforms. All systems created

[9] https://www.docker.com/
[10] https://kubernetes.io/

a *Vertical Service* entity that encapsulated all information related to a given E2E network service. This led to my decision to also implementing a *Vertical Service* abstraction in the POC.

The negative aspects gathered from the three projects were the main motivation for the POC developed. All three orchestration systems followed a monolithic architecture, two of them had non-standard network slicing support, and two of them had limited inter-domain support. Being those the three main problems of the three current SoA vertical service orchestrators, this thesis POC aimed at solving them.

CHAPTER 4

Proof-of-Concept

After analyzing all the State of the Art vertical service orchestration solutions, and gathering their advantages and disadvantages, I concluded that some problems spanned across multiple platforms, and for that reason, needed to be addressed. A POC platform was developed, motivated by solving the core problems encountered. Those problems are:

1. Monolithic architectures
2. Non-standardized network slicing
3. Inadequate multi-domain support

Briefly explaining why those aspects are problems: concerning the architecture, although modular, all SoA solutions are a unified monolithic system composed of several Java Spring-boot[1] applications. Monolithic applications are not the best solution for the maintainability, flexibility, and scalability needed for this kind of orchestration system. Regarding the non-standardized network slicing support, only the 5GR project, through an add-on, supports network slicing according to current standards, which demanded workarounds and may have impacted the final system's quality. This issue is quite relevant since by not supporting the current standards, the interoperability and integrability decrease significantly. Finally, the inadequate multi-domain support problem is similar to the last one, where only the 5GR project has a more mature solution but is achieved again with an add-on. Once again, this issue is relevant, since by not correctly supporting inter-domain scenarios, vertical use cases with higher complexity that demand such functionality cannot be supported.

4.1 System Architecture

The final system, named NetOr, consists of an OSS/BSS system that operates over the operator's 5G infrastructures and services. Platforms of this style allow abstracting the both the intricate actions needed to deploy a network service, and the infrastructure and network complexities. With those abstractions in place, the end-user(vertical industry) can concentrate

[1] https://spring.io/projects/spring-boot

its efforts in developing and analyzing the services and functions needed for the organization's objectives.

NetOr follows a micro-service and event-driven architecture. Each managed entity in the system is handled by a unique component, which communicates with others through a centralized message bus to exchange event messages. With this architecture, the system is more scalable, flexible, modular, and efficient.

Distinct and isolated components compose the system, enabling it to scale a unique micro-service if it is the bottleneck in the overloaded platform, easily creating more workers to manage and handle that specific set of operations and entities. That was possible by having each micro-service as a stateless component, meaning that they don't keep any runtime information. For that reason, when a microservice needed to store runtime data, it used a memory cache to guarantee that all data was persisted at all times, also serving as a fault tolerant mechanism capable og handling system crashes.

Flexibility is another characteristic of the system, achieved once again by having separate and independent components. With each micro-service having a robust communication interface aligned with the most recent standards, it allows a seamless substitution of any sub-component, given that the new component provides the same functionalities anf follows the same interface and standards. By having this type of flexibility, the internal implementation of any micro-service can be quickly replaced and updated, without restrictions over the language or technologies used. This substitution can occur without interfering with the remaining system.

Modularity, although similar to flexibility, focus on allowing easy and effortless addition of new components. With a centralized message bus containing all published events, a new microservice can use that information and add new functionalities to the system without impacting the remaining platform. Even if the new component interacts with others, given that the environment is as decoupled as possible, the changes needed are minimal. In addition, modularity also facilitates removing any micro-service, assuring the modifications to the system are as minimal as possible.

By default, a micro-service oriented system favors asynchronous communications. In comparison with a sequential approach, asynchronous communications allow parallel processing, which may in some cases improve the performance and efficiency of a given process. For that reason, this system may be able to improve the performance over the execution of services management operations when compared to other already existing similar orchestration systems. on the other hand, this type of architecture has its own drawbacks, being one of them the need to exchange considerably more messages through the network when compared to monolithic systems. By needing to constantly communicate to exchange information and the system status between the micro-services, this may increase the platforms performance delays.

Finally, another requisite for the NetOr system was to facilitate its interaction with verticals. For that reason, a web portal was developed to help the vertical user to interact with the NetOr system. Through a minimal interface, the portal presents the available actions in the most intuitive way possible and abstracts as much as possible the underlying complexities.

The final NetOr architecture is the one presented in 4.1.

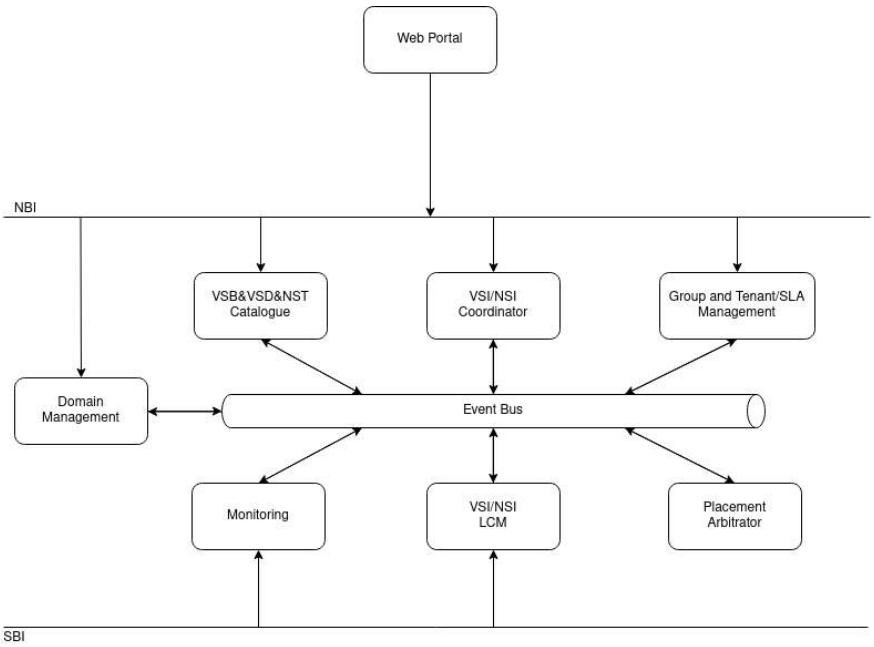


Figure 4.1: High level proposed architecture

4.2 Network Orchestrator(NetOr)

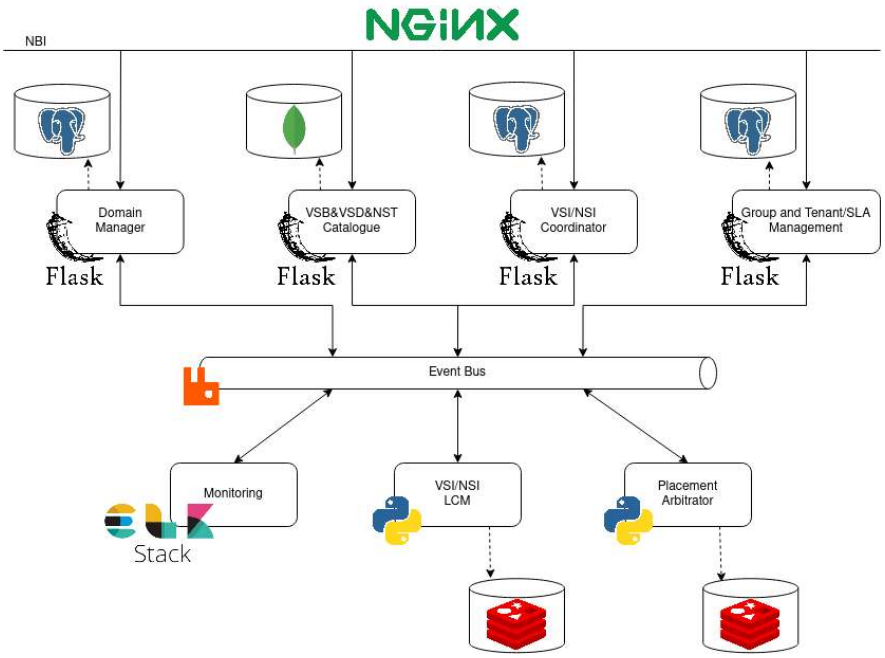


Figure 4.2: Final system architecture with associated technologies

As presented in Figure 4.2, the system is composed of 6 main components: the VSB/VSD/NST Catalogue, the Group/Tenant Manager, the Domain Manager, the VSI/NSI

Coordinator, the VSI/NSI LCM Manager, and the Placement Arbitrator. As per micro-service-oriented systems, each sub-component is responsible and manages a different set of entities or a different set of functionalities. The following points present a description for each component, technologies used, dependencies, and interactions with other elements. All micro-services have Python[2] as their base technology, which was never used for this kind of system according to the state-of-the-art solutions.

- **VSB/VSD/NST Catalogue**: This component serves as a persistence service, allowing the creation and onboarding of new descriptors and templates. It also enables the management and deletion of those resources. Its main objective is to provide a centralized entity capable of storing and supplying all the needed descriptors and templates related to vertical services. Those entities are the VSBs, VSDs, NSTs, NSDs, and VNFs. The main entity is the VSB, which serves as a template for the vertical service, defining the topology, QoS parameters, policies, and lifecycle aspects. These blueprints are created by operators to define and expose the services supported by them. Verticals, based on those blueprints, can create VSDs, which are a specification of the predefined service structure, specifying the QoS parameters their scenario requires. Since the vertical services can be composed by network slices or network services, their definitions can also be onboarded in the system, whence the support of NSTs, NSDs, and VNFs. To provide the creation, deletion and management operations needed, this component has a REpresentational State Transfer (REST) API, which is implemented using the well-known Flask[3] library and documented using the Flasgger[4] library that creates a Swagger/OpenApi[5] documentation page.

  As its dependencies, this component needs a MongoDB[6] to persist all managed entities. To select this database technology, I performed an extensive analysis of the resources to store, their purpose, and how other components would use them, concluding that a document-based database was ideal since all descriptors and templates when onboarded did not suffer alterations. If they need modifications, the end-user should replace the entire descriptor by deleting it and creating a new version, which allows a document-based persistence, and simplifies the data model and management aspects. This component must also interact with the centralized RabbitMQ[7] message bus to receive and send the necessary information. I used the Python Pika[8] library for this integration. This component also interacts with a centralized IdP system, in this case, implemented in the Group/Tenant Manager detailed in the following points.

  Finally, as proof that the micro-service architecture improves the system in runtime, production, and development phases, another person developed this component in parallel with me developing the remaining micro-services. That proves that, since

[2]https://www.python.org/
[3]https://flask.palletsprojects.com
[4]https://www.flasgger.org/
[5]https://swagger.io/
[6]https://www.mongodb.com/
[7]https://www.rabbitmq.com/
[8]https://pika.readthedocs.io/en/stable/

each microservice is independent, isolated, and as loosely coupled as possible, their development can be made in parallel, needing only to establish the information model exchanged between components.

- **Group/Tenant Manager**: This component is responsible for handling both the Groups and Tenants of the system, allowing the creation, management, and deletion of both. It serves not only as a persistence service that stores and provides those two entities, but it is also an IdP for the rest of the system. This component provides a centralized authentication service for all the system's users, checking if the tenant exists and if the password matches with the one stored in the system. For that, this component implements an Oauth2[9] authentication service, achieved using the Flask-OAuthlib[10] library. Oauth2 was select since it is the industry-standard protocol for authorization. This component also exposes a REST API, implemented using the well-known Flask library and documented using the Flasgger library that creates a Swagger/OpenApi documentation page.

  In terms of dependencies and interactions, this component depends on a relational database to persist the managed entities, which proved to be the best option after a detailed analysis of the entities and information needed to be stored. That was the case because the relations between entities were relevant and data was expected to be changed(change tenant name), being a relational data model the most adequate solution. I chose a PostgreSQL[11] instance for this POC since it is one of the most used Database Management System (DBMS), with an easy installation and a simple integration with the technologies used in this project. To achieve this integration, I used the SQLAlchemy[12] library. In addition, this component must also interact with the centralized RabbitMQ message bus to receive and send the necessary information. I used the Pika library for this integration.
- **Domain Manager**: This component is responsible for managing everything domain-related. Besides the creation, management, and deletion of domains, this microservice also handles communications with the lower-level orchestration entities responsible for managing each domain. This component was developed in a modular way, enabling the possibility of having multiple drivers and easily adding new ones. This allows the NetOr to be technology agnostic, enabling its communication with different orchestration technologies, from NFVOs to SDN controller if needed. This component also needs to expose a REST API, implemented using the well-known Flask library and documented using the Flasgger library that creates a Swagger/OpenApi documentation page.

  In terms of dependencies and interactions with other components, this module needs a PostgreSQL database to store all the information related to the existing domains. I achieved this integration using the SQLAlchemy library. This component must interact with the centralized RabbitMQ message bus to receive and send necessary information

[9] https://oauth.net/2/
[10] https://flask-oauthlib.readthedocs.io
[11] https://www.postgresql.org/
[12] https://www.sqlalchemy.org/

and with the centralized NetOr IdP. I used the Pika library to achieve the integration with the RabbitMQ message bus.

- **VSI/NSI Coordinator**: This component is responsible for handling all operations related to the high-level VSI, such as its creation, execution of operations during runtime, termination, and deletion. This component is the one responsible for triggering the VSI orchestration processes for the rest of the system. It is this component that manages the record associated with each VSI, meaning that this coordinator has all the information about all vertical services, such as their status. This component exposes a REST API, implemented using Flask and documented Flasgger.

  In terms of dependencies and interactions, this module needs a PostgreSQL database to store all the information related to the existing VSIs and its composing NSIs and NSs. I achieved this integration using the SQLAlchemy library. This component also interacts with the centralized RabbitMQ message bus to receive and send necessary information and with the NetOr centralized IdP system. I used the Pika library to achieve the integration with the RabbitMQ message bus.
- **VSI/NSI LCM Manager**: This component is responsible for managing and coordinating the VSI themselves and possible sub-components, such as NSIs and NSs. For this, each VSI originates a new management agent that will handle all operations needed to be made for that vertical service. It is that agent that will manage the entire lifecycle of the VSI, and guarantee that the NetOr system has always the most recent information about the service, updating its status regularly.

  In terms of dependencies and interactions, this module needs a Redis[13] instance as a memory cache. Redis was chosen because it is one known and used technology for this purpose. This cache serves to persist all relevant runtime information about VSIs, NSIs, and NSs. To communicate with that memory cache I used the Python Redis[14] library. This component must also interact with the centralized RabbitMQ message bus to receive and send necessary information and with the NetOr centralized IdP system. I used the Pika library to achieve the integration with the RabbitMQ message bus.
- **Placement Arbitrator**: This component is responsible for processing all the information related to Vertical Services, such as blueprints, descriptors, and templates that configure it to generate the placement information needed, defining the deployment location of each sub-component and possible restrictions. Additionally, it also considers SLAs associated with the tenant and parameters dynamically defined during instantiation. By processing all that information, this component can also arbitrate over the VSI and its sub-components. For this, each VSI originates a new agent that will process that information, generate the placement directives, and arbitrate over it, if necessary.

  In terms of dependencies and interactions with other components, this module also depends on a Redis instance to persist all runtime information about NSIs, NSIs, and NSs in a memory cache. I achieved this integration by using the Python Redis library.

[13] https://redis.io/
[14] https://pypi.org/project/redis/

This component must also interact with the centralized RabbitMQ message bus to receive and send necessary information and with the NetOr centralized IdP system. I used the Pika library to achieve the integration with the RabbitMQ message bus.

4.2.1 APIs and Data Models

Concerning the norms and standards supported by the NetOr system, those previously presented in Section 2.5 heavily influenced my selection. There are two distinct areas where the NetOr system should consider and support standards and norms, which are the interfaces interconnecting the NetOr with external components, namely its NBI and SBI. The NBI should follow well-known community accepted standards to enable the system to be easily adoption by third-party platforms, to allow the seamless substitution with equivalent platforms, and to guarantee that all required parameters for the underlying systems are present in the data models. Similarly, it is also crucial that the NetOr's SBI, the interface interacting with underlying orchestrators, such as NFVOs, follows well-defined and accepted standards for various reasons, such as allowing the support of different orchestrator technologies, such as distinct NFVOs. Another reason for supporting such norms is that the parameters needed for the resources and features of those NFVOs are, by default, correctly detailed in the data models.

In terms of the NBI, after analyzing the most relevant standards of the area, I selected several, most of them related to the data model exchanged in Create, Read, Update and Delete (CRUD) operations related to VSIs and auxiliary resources. Those standards are TS 28.541[11], SOL005[13], and SOL006[14], all mainly focused on information models. This standards define the NSDs, VNFs, and NSTs data structures, and helped understanding which parameters the VSBs and VSDs should support. In addition to that, I also based the data structures adopted by the NetOr POC on the models adopted by the most mature SoA vertical service orchestration system, which is the 5GR-VS. As expected, these models needed to be extended due to the new functionalities supported by NetOr. Those extensions focused mainly on the VSB, VSB-related entities, and in the VSI instantiation request data models. Since NetOr provides service modification support through the usage of runtime primitives, the VSB needed to be extended to contemplate the actions supported by a given VSI. Similarly, it was necessary to define the data structure where the service's supported actions were detailed, describing its id, name, parameters, and possible default values. The VSI instantiation data model was also extended because NetOr added the support of dynamic instantiation configurations and domain deployment selection. For that reason, I added the option of defining the target component and its instantiation configuration, as well as the option to define the deployment domain of each vertical service component.

Concerning the SBI, it was also necessary to analyze the most relevant standards of the area and select the best-suited ones, focusing on data models and operations. This standard selection process is crucial for this interface since it handles the interaction with well-known NFVO platforms, assuring that if the correct norms are selected, it will increase the NetOr platform compatibility with more NFVO systems. The chosen standards are TS 28.530[10],

TS 28.531[9], TS 28.541[11], TS 28.801[6], SOL005[13], and SOL006[14]. These norms helped define which operations needed to be supported, such the operations related to NSs and NSIs, as well as the data structure each request should follow when communicating with the underlying NFVOs. Although those standards have a considerable level of detail and are complete concerning the operations needed for the NetOr platform, they needed an extension. In particular, it was necessary to add management functions that allowed the execution of runtime operations over instantiated network services and network slices. Additionally, the models and actions adopted for the SBI of the most mature SoA vertical service orchestration system, which is the 5GR-VS, also influenced NetOr's operations and data structures.

4.3 Multidomain Automatic Mechanism

Concerning mobile service and network coverage, different operators manage distinct geographic zones, limiting the areas where each operator can provide their services directly. If a client of a given operator enters a region covered by another operator, there are mechanisms to enable the redirection of that user's requests to the correct operator. Currently, the approach when instantiating Vertical Service, Network Services, and Network Slices assumes that a unique domain should provide them, redirecting to it the data and requests of an end-user if he enters an area covered by another operator. It is easy to imagine many scenarios where these redirecting mechanisms could impact the final E2E service, such as automotive or medical scenarios that rely on very low latency. In those scenarios, the ideal solution is to have the service in question instantiated across the various domains with which the end-user may interact. With this in mind, the inter-domain mechanism appears to enable the on-demand connection of independent regions.

This mechanism is a recent innovation in the NFV and service orchestration world. It is a powerful functionality that disrupts the current modus operandi of network and service providers. Currently, those providers follow guidelines and restrictions that force their clients to be only served by them, meaning that all the service and network functions are hosted and instantiated in their unique domain. Because of the need to maintain all services and resources in one area and controlled by one operator, mechanisms such as roaming appeared to guarantee those restrictions even if the user enters a domain covered by another operator. The closest approach to inter-domain that operations can achieve under those restrictions is federation. Federation interconnects multiple areas but not in a plug-and-play fashion, needing the establishment of a priori contracts to enable the connection of those domains, defining the interactions, resources, and services that may be exchanged or used between them.

Similar to the federation mechanism, the multidomain innovation aims at creating an improved environment and infrastructure for services and applications through the connection of multiple independent domains, which will increase the benefits to users of individual operators. But when compared with the federation mechanism, the inter-domain feature intends to achieve this without defining a contract and restricting the interaction of the

independent domains. By using standards for operations, APIs, and data models, which many of the domain's platforms should already support, this mechanism is easier to achieve.

Several scientific works, and even recent standards, are starting to study and propose best practices and architecture options when dealing with this inter-domain scenario. An example of standards that are starting to consider this scenario is TS 28.801[6], which suggested three different solutions for multiple operator coordination management. The first approach is the instantiation of a NSI-based communication service across multiple operators, relying on the customer owning the CSMF and communicating with the various operators, which must provide NSMFs. A second option is an operator management system to management system interaction, meaning that the customer will communicate with a main operator hosting the CSMF, which in turn will interact with its and the other operator's NSMFs. Finally, a third option is an NSMF to NSSMF approach, meaning that the customer will communicate with a main operator hosting the CSMF, that entity will delegate the needed action to its domain's NSMF, which will leverage the multiple operators functionality by interacting with its and the other operator's NSSMFs.

An example of a scientific article analyzing this problem is [30], which proposes a 5G vertical service orchestration framework focused on network slicing on an inter-domain scenario. The paper starts by analyzing the entities required for inter-domain orchestration, such as the Vertical/Digital Service Consumer (DSC) that uses the services provided by the Digital Service Providers (DSPs), which in turn are dependent on the capabilities and resources provided by the Network Service Providers (NSPs). The DSP is responsible for managing the service lifecycle and its exposition to Verticals. The DSC only consumes the services exposed by the DSP, needing only to fill the QoS parameters of said services.

The article's authors also present the main challenges when dealing with E2E NSIs orchestration in an inter-domain scenario. The first one is that the DSP needs to give detailed information about the Network Slices supported by the different NSPs, meaning that well-defined communication protocols should exist. The other challenge is to deal with the E2E NSI provisioning process, since during the instantiation, configuration, and runtime phases of the NSIs composing the E2E NSI, the DSP orchestration platform needs to regularly receive new information about them, which once again, require well-defined communication and monitoring APIs. Additionally, the performance information obtained by those updates is crucial to the operation of this inter-domain orchestrator, since the fulfillment of pre-defined SLAs is mandatory, possibly requiring reactionary actions over the NSIs.

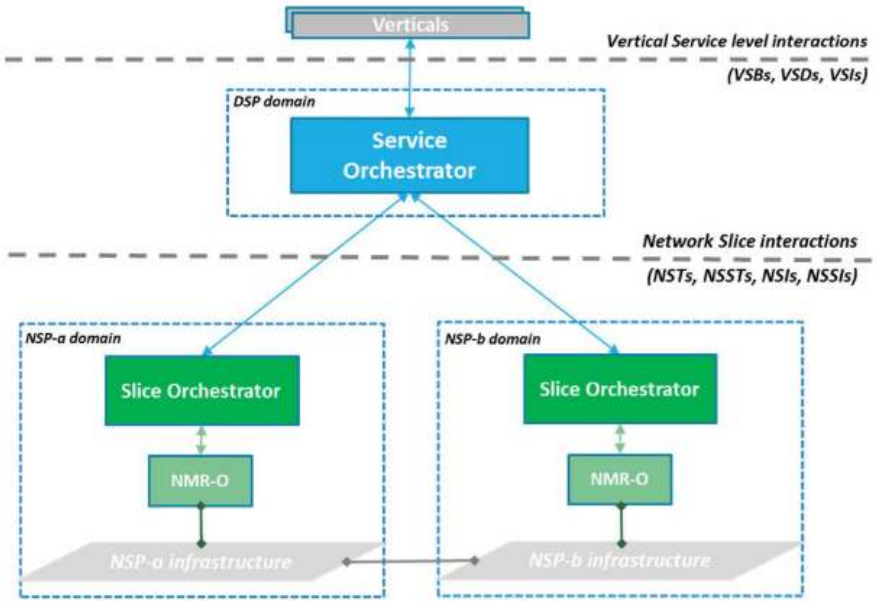


Figure 4.3: Multi-domain orchestration high level architecture [30]

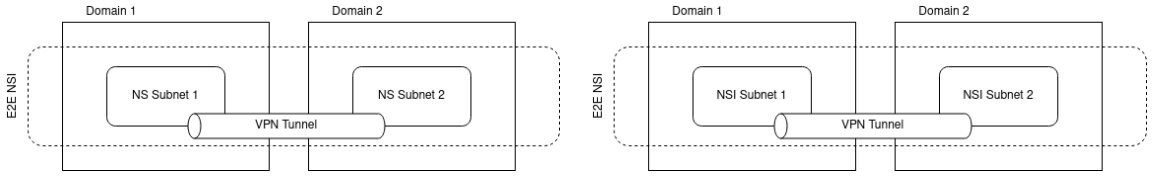
The article then presents the proposed inter-domain orchestration framework, which follows the architecture depicted in Figure 4.3. It has three levels of orchestration logic: the higher one, located in the DSP layer, is the Service Orchestrator and it's responsible for the vertical services; the second one, located in the DSP layer, is the Slice Orchestrator and it's responsible for the Network Slices; the third level, also located in the NSP layer, is the Resource Orchestrator and it is responsible for the slice resources. The paper also presents the data models used to achieve this inter-domain orchestration, relying on VSBs and VSDs to define vertical services, in NSTs to describe NSIs, and in NSDs and VNFDs to configure the network slice resources.

The chosen architecture for this POC follows the one proposed in the first multi-operator coordination solution presented in TS 28.801[6] and the architecture proposed in [30]. Both architectures resemble the same, attributing different names to similar entities. Both architectures propose a centralized service orchestration agent(CSMF/Service Orchestrators) that will communicate with lower-level orchestrators(NSMFs/Slice Orchestrator). The improvement of the mechanism developed for this thesis' POC is the definition of a solution to connect the independent E2E Network Slice Subnets, which was not studied in both documents. The proposal is to create and configure a Virtual Private Network (VPN) tunnel between them, creating a secure communication channel between domains. For this POC and use cases, the technology used to implement this VPN tunnel was Wireguard[15], a straightforward and minimalist framework that allows the rapid instantiation and configuration of VPNs. Additionally, this POC itself is also an improvement over the mentioned related works, since non of them validated their proposed architectures.

Concerning this mechanism, the main objective in this POC is to instantiate an E2E Service across multiple domains without prior negotiations. That is technically possible by

[15]https://www.wireguard.com/

using an E2E Network Slice composed of Network Services or Network Slices. There are three distinct options for the E2E Network Slice: its subnets are composed of NSs; its subnets are composed of other Network Slices; and finally, there is the hybrid approach. Each of those approaches can generate the same Network Slice architecture, where the differing factor is the abstraction level of the entities deployed at each domain. To better understand some of the possible architectures of that E2E Network Slice, Figure 4.4 presents the NS and the NSI approach.



(a) NS composed E2E NSI architecture **(b)** NSI composed E2E NSI architecture

Figure 4.4: E2E NSI possible architecture

To successfully achieve the inter-domain, not only do the network resources (NSs and NSIs) need to be aligned with this mechanism, the centralized service orchestrating agent should exist, ideally outside all domains in question, to allow the connection of the tunnel peers. That agent will receive and process the dynamic tunnel endpoints' information and exchange it with the remaining peers, effectively completing the tunnel configuration. NetOr, in addition to all the other features and functionalities, also serves as that centralized service orchestrating agent, gathering and redistributing the inter-domain information. Figure 4.5 presents the expected data flow between the NetOr and the various domains.

As presented in Figure 4.5, the process starts by instantiating a Vertical Service based on blueprints, descriptors, and templates that support and activate the inter-domain mechanism. With that request, the system creates all necessary management entities and instantiates the network resources(NSIs or NSs) in the respective domains. After instantiating the Vertical Service, there is a continuous polling over the status and information of its components. When the NetOr system verifies that a Vertical Service component is "running" (meaning that it is deployed and configured), it triggers a runtime operation to fetch the tunnel information, such as the IP of the tunnel peer machine and the self-generated tunnel public key. Once all the Vertical Service components are "running" and their tunnel information have been fetched, the NetOr proceeds to exchange that information between peers, effectively providing the necessary data to peers and configuring the tunnel in the process. Only the NetOr knows all tunnel peers since it instantiated them independently on different domains.

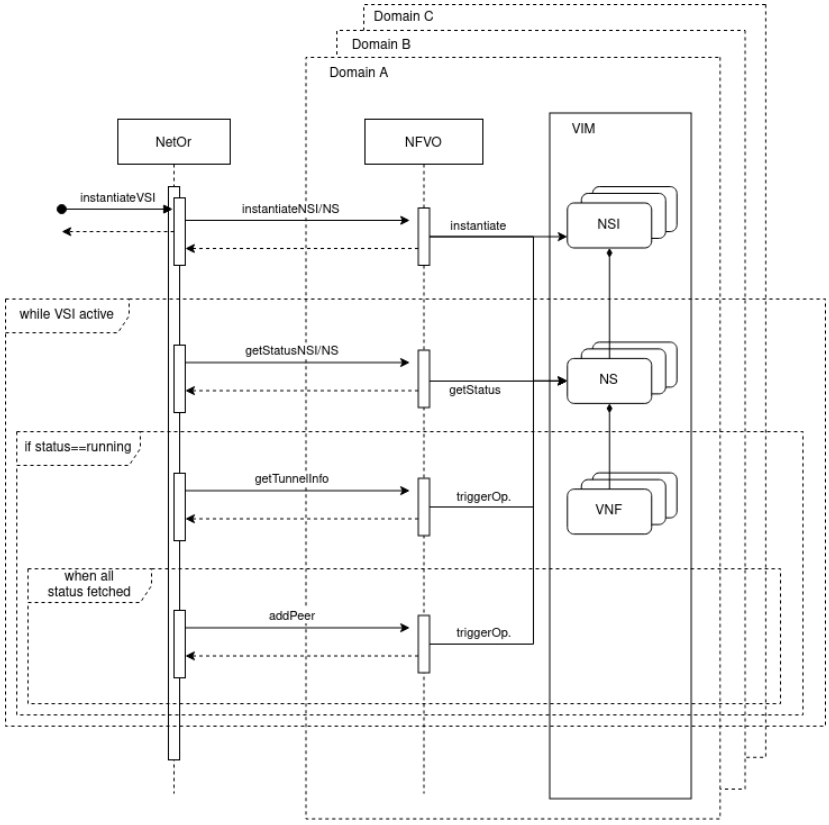


Figure 4.5: Data Flow between NetOr and respective Domains

Novel Approach

Another approach envisioned, designed, and implemented in the early stages of the NetOr project proved to be a novel option and didn't align in some standards and existing system architectures. Current systems and the overall community defends a specific hierarchy of communication in the NFV environment, meaning that a given layer of abstraction may only communicate with the two neighboring ones, one above and one below. An example of this is, in the inter-domain scenario where there is an NSI composed of two NSs, each comprised by one VNF, the NSI should only interact with the NSs, a NS may only interact with the NSI and the VNF, and the VNF should only interact with the NS.

This novel solution breaks that hierarchy by suggesting that, if successfully ending the instantiation and basic configuration phases, which effectively defines that the service is ready to be used, the tunnel peer VNF would send its tunnel information to an already known entity that will handle the inter-domain logic. The data flow expected from this novel approach is the one presented in Figure 4.6

Although different from already existing systems and standards, this approach can have some interesting characteristics that can improve the overall service, such as decreasing the delay in receiving the inter-domain tunnel information. Since the system doesn't need to wait for the polling action, depending on the implementation, the instantiation time of an inter-domain service can reduce significantly. Another benefit is the overall mechanism

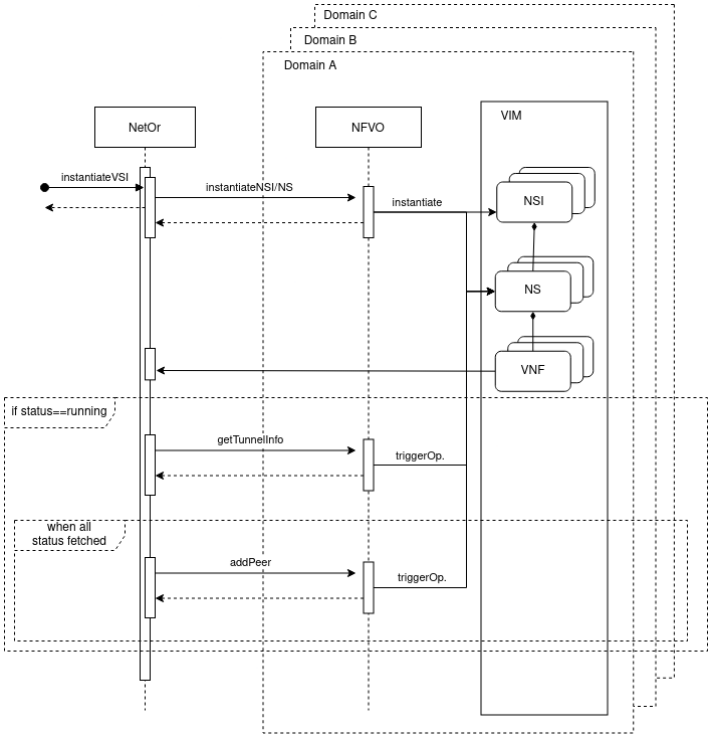


Figure 4.6: Novel Approach Data Flow between NetOr and respective Domains

simplification since the end of configuration event is automatically propagated, disabling the need to continuously query the NS or NSI about their status and consequently fetching the tunnel information.

4.4 Web Portal

In addition to the backend service developed to support all OSS/BSS operations and the verticals' logic, I developed a web portal user interface to create an all-rounded complete system. The reason for choosing a web portal over other options, such as dedicated applications, is that an interface like this is always accessible from anywhere, needing only an internet connection. This approach also simplifies updating the portal, since if hosted in a public server, updating it is as simple as upgrading the version in that server, making all clients immediately start using the new version. When updating and availability is a paramount system requirement, there is also the possibility to create a secondary parallel production server, needing only to redirect the Uniform Resource Locator (URL) used for the portal to the new upgraded portal. These possibilities are all possible due to the portal being web-based.

The portal objective is to facilitate the vertical interaction with the NetOr system by defining a graphical interface. Additionally, the portal supports two distinct user profiles, one for an admin user supposed to have extensive knowledge about the platform and underlying technologies, and another for a tenant user that may know less about the technologies involved. When using this platform, those profiles have different objectives: the admin aims at managing groups, tenants, domains, and auxiliary service structures; the tenant's goal is to use those

pre-defined structures to instantiate services for his use cases. The actions provided by the portal to each of those profiles are:

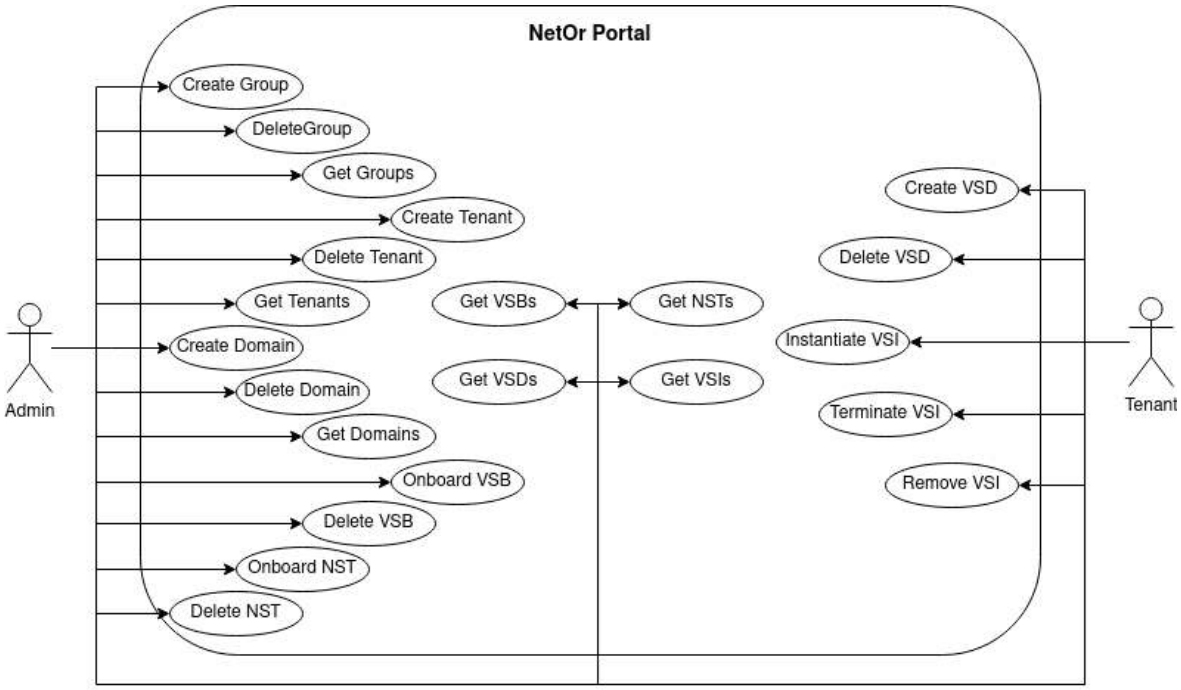


Figure 4.7: Portal's use case diagram

The main action pages for both the admin and tenant profiles are presented in Appendix A.

4.5 Deployment

From the beginning of the platform development, there was an emphasis on guaranteeing that the system could run in any machine, operating system, and environment possible. This approach avoided from the beginning typical problems, such as bugs and errors when deploying the system in new VMs or Operating Systems (OSs).

From my knowledge, the best option to achieve this interoperability is to use containerization tools, such as Docker[16]. This technology uses OS-level virtualization to deliver software packages called containers, which are isolated from one another and bundle their software, libraries, and configuration files. Although independent and isolated, containers can communicate with each other through well-defined channels. Lastly, given that all containers share the services of a single OS kernel, they end up using fewer resources than virtual machines. In conclusion, Docker provides a middle layer between my system and the OS, assuring that if a given environment supports Docker, NetOr will deploy and function correctly.

There are several approaches when containerizing a given system, and the most straightforward one is to encapsulate the entire platform in a container and provide it as a service. This option is the most adequate when dealing with monolithic systems impossible to subdivide or

[16]https://www.docker.com

a project where scalability and modularity are not architectural requirements that need to be assured. When dealing with systems with micro-service architectures or distributed by nature, another option is to containerize each sub-component of the system, meaning that the system is composed of a group of containers and not a unique one. These container groups, if handled cohesively, are also known as a stack in Docker.

This last approach, where a system contains several containers, has numerous advantages, mainly when debugging, maintaining, and scaling the system. With isolated containers, if there is a system error during the execution of an operation, it can be easier to find the source of the problem since each container provides its logs in an isolated manner. Additionally, it is easier to maintain this system since only the problematic component needs correction when dealing with errors; similarly, updating a sub-component can be independent and does not interfere with the remaining system, enabled because the containers are isolated.

Lastly, concerning scalability, since all containers are isolated and separated, each one can be scaled up or down as needed. Furthermore, using this deployment approach enables the future exchange of the container orchestrator, opting for a better and more reliable one, such as Kubernetes[17]. That orchestration technology is better than the simple Docker orchestrator since it has more features and better container stack management, mapping each container to services and allowing their automatic scaling by increasing or removing container instances, as necessary. That is paramount for a system that is in a production environment and may need to scale certain portions of the platform to continue providing a good service.

For this prototype, I enabled this deployment approach by creating a Dockerfile in each component of the system, which configured and defined that sub-component container, the libraries needed, and the services that would run inside it. To ease the management of those containers and reduce the actions performed to enable all modules, I used Docker-Compose to define and configure a stack of containers in a single YAML file, interacting with it as a single entity.

4.6 NFVO Resources

There must be a mutual and coordinated effort from both the orchestration and resources layers to achieve the inter-domain functionality. Upper orchestration and management layers have functions and entities responsible for high-level resources, such as E2E slices and services. The lower resource layers refer to underlying platforms, infrastructures, and networks that also need to support the inter-domain mechanism. That support may include assuring the new resources and entities support the inter-domain feature, as well as guaranteeing the NFVOs and VIMs are configured appropriately.

With that in mind, it was necessary to create new custom entities, such as new VNFs, NSs, and NSTs, since this inter-domain approach is new. The NFVO used for this POC was OSM Release 9, and for that reason, all the developed entities follow the guidelines, requirements, and standards defined in its documentation. In this OSM release, the VNFDs and NSDs follow the SOL006[14] data models, and the NSTs follow the SOL005[13] information structures.

[17] https://kubernetes.io

4.6.1 VNFs

The VNF in the NFV environment is the entity responsible for virtualizing and implementing the network functions. It is in the VNFs that specific behaviors or functionalities are implemented so that future network services can leverage them. To configure such entities, a VNFD was created to define all parameters and lifecycle management aspects related to them. In the OSM platform, the VNFD is a package composed of several crucial components: a YAML Ain't Markup Language (YAML)[18] configuration file aligned with the SOL006[14] data models and containing the information used by the NFVO; a Juju charm[19], adopted to enable the customization of the VNF lifecycle management stages; and finally, Cloud-init[20] configuration files, adopted to enable a quick configuration of common properties of the VNF's VMs. Furthermore, OSM also specifies that the VNFD package should aim to fullfil the lifecycle stages required for the correct operation of the function, which are basic instantiation, service initialization, and runtime operations, also known as Day-0, Day-1, and Day-2 operations, respectively[31].

For this new inter-domain mechanism approach, I created a new VNF package with special efforts in the YAML descriptor file and the Juju charm. The development of this new VNF had special attention in ensuring the fulfillment of required lifecycle stages. The Day-0 stage relies on the YAML descriptor and the Cloud-init files to deploy and initialize the function's VM topology. With that in mind, I guaranteed that all my configurations were correct, resulting in a VNF with the topology presented in Figure 4.8.

As shown in Figure 4.8, the VNF internal topology has only one VM, also referred to as Virtual Deployment Unit (VDU) by the OSM documentation, with two network interfaces. Those interfaces are in turn mapped to VNF interfaces and exposed to the exterior. Below, I present a portion of the YAML configuration file containing the definition of said topology, alongside other characteristics (Cloud-init file and Juju charm associated). In this case, the cloud-init configuration file defines passwords and static definitions, such as enabling authorization in ssh.

[18] https://yaml.org/
[19] https://jaas.ai/
[20] https://cloud-init.io/

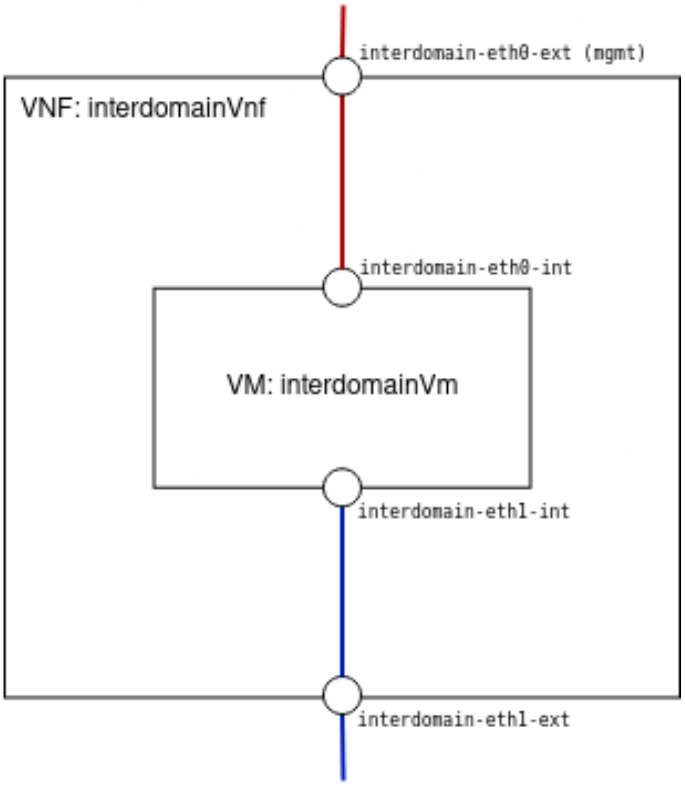


Figure 4.8: Internal interdomain VNF topology

```
vnfd:
  ext-cpd:
  - id: interdomain-eth0-ext
    int-cpd:
      cpd: interdomain-eth0-int
      vdu-id: interdomain
  - id: interdomain-eth1-ext
    int-cpd:
      cpd: interdomain-eth1-int
      vdu-id: interdomain
  id: interdomain-vnf
  mgmt-cp: interdomain-eth0-ext
  vdu:
  - cloud-init-file: cloud-init
    id: interdomain
    int-cpd:
    - id: interdomain-eth0-int
      virtual-network-interface-requirement:
      - name: eth0
        virtual-interface:
          type: PARAVIRT
```

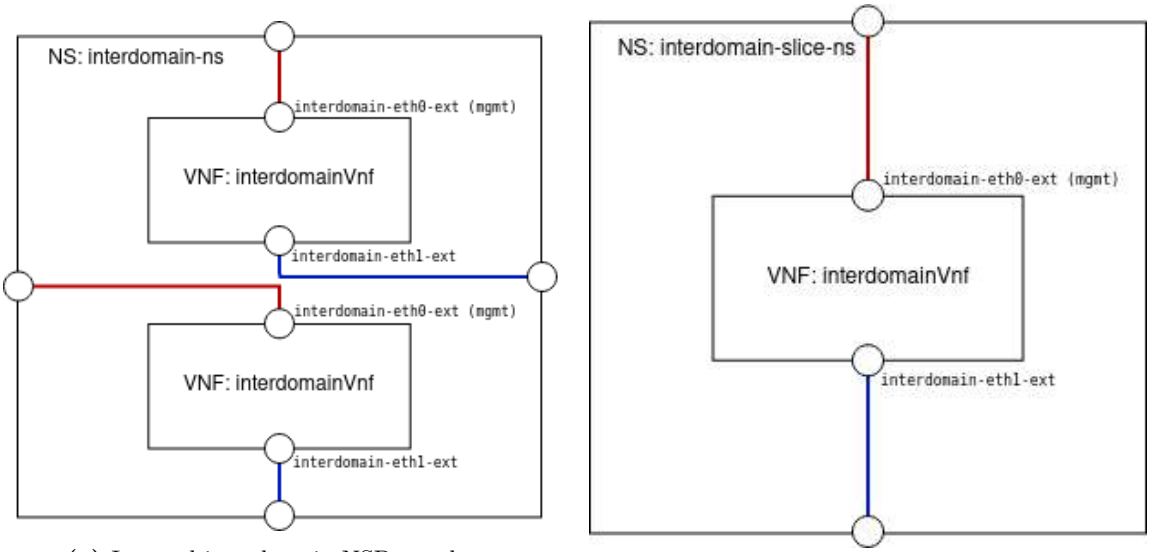
```
    - id: interdomain-eth1-int
      virtual-network-interface-requirement:
      - name: eth1
        virtual-interface:
          type: PARAVIRT
    sw-image-desc: ubuntu18.04
    virtual-compute-desc: interdomain-compute
    virtual-storage-desc:
    - interdomain-storage
  vnf-configuration:
  - config-primitive:
    juju:
      charm: interdomainvdu
...
```

After going through the basic initialization phase, the VNF executes Day-1 and Day-2 operations. The Juju charm was adopted by OSM to enable the easy customization of these two configuration lifecycle stages. Concerning this POC and the inter-domain scenario, I created a charm where during the Day-1 phase, it installed libraries and tools needed for the creation and modification of the VPN tunnel. That Juju charm also defined four possible Day-2 operations: an action to add new tunnel peers, an action to remove existing tunnel peers, an operation to retrieve the VNF's information (such as endpoint and tunnel public key), and an operation to modify the tunnel performance, particularly its bandwidth. In the Juju charm, both Day-1 and Day-2 operations are implemented in the same way, needing to differentiate them in the YAML configuration file, as defined by OSM [31].

4.6.2 NSs

With functional VNFs, it is the NS that operationalizes them. NSs are composed by one or more VNFs, effectively combining them and providing a cohesive service. To configure those services, NSDs were created to specify which VNFs should exist in the service, and how they should be connected. In the OSM system, those NSDs are packages, where the most important component is the SOL006[14] aligned YAML configuration file. For this prototype's inter-domain scenario, I created two distinct NSDs, both to support the inter-domain mechanism. The difference between them is the number of VNFs used: one defines two independent VNFs acting as VPN endpoints, developed for testing the inter-domain mechanism in a service level; the second NSD defines a unique VNF, designed for testing the inter-domain mechanism in the network slicing level. The VNFD used for all NSDs was the same, and previously described in subsection 4.6.1. Figure 4.9 presents both topologies to help illustrate their implementations and differences.

Below I present an example of the descriptors developed, specifically a portion of the NSD created for the network slicing level inter-domain support. As already mentioned, the descriptor defines one independent VNF and the networks connected to it.



(a) Internal interdomain NSD topology

(b) Internal interdomain slice NSD topology

Figure 4.9: Topologies of both NSDs created

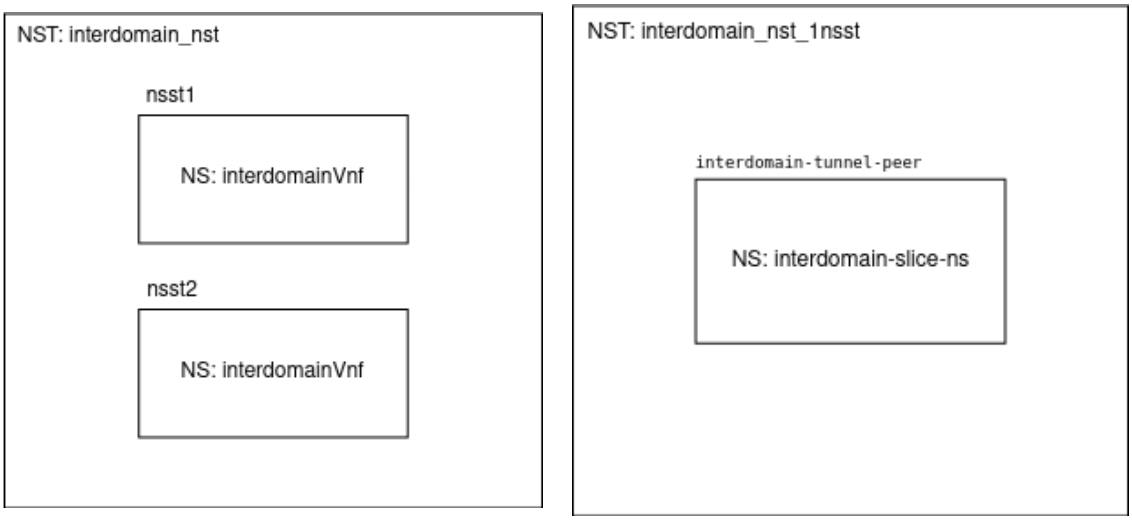
```
nsd:
  nsd:
    df:
    - id: default-df
      vnf-profile:
      - id: '1'
        virtual-link-connectivity:
        - constituent-cpd-id:
          - constituent-base-element-id: '1'
            constituent-cpd-id: interdomain-eth0-ext
          virtual-link-profile-id: mgmtnet
        - constituent-cpd-id:
          - constituent-base-element-id: '1'
            constituent-cpd-id: interdomain-eth1-ext
          virtual-link-profile-id: datanet
        vnfd-id: interdomain-vnf
    id: interdomain-ns
    virtual-link-desc:
    - id: mgmtnet
      mgmt-network: true
      vim-network-name: external
    - id: datanet
      vim-network-name: test
    vnfd-id:
    - interdomain-vnf
```

...

4.6.3 Network Slices

Finally, there are Network Slices, which are the highest-level entity in the OSM framework when dealing with network resources. These entities follow the definitions established in TS 28.801[6] and are mapped into the ETSI-NFV MANO architecture through the mapping presented in NFV-EVE 012[12]. Network slices' intended use is to enable 5G use cases and provide isolated specialized networks for different purposes. In addition to that, since each network slice is isolated, that opens the possibility to provide distinct service quality profiles. For that reason, there are three main service quality profiles, and each network slice commonly follows one of them. As implemented by OSM, Network Slices in practice operate as a particular kind of Network Service or, more generally, as a set of various Network Services handled as a single entity. The Network Slice contains one or more subnets, which in turn are composed of network services. To configure such slices, NSTs were created as a deployment template containing all the information needed by the NFVO. In OSM, these templates are YAML configuration files following the data model defined in SOL005 [13].

For this prototype, I created two distinct NSTs for testing the inter-domain mechanism in different scenarios, one for a single domain and another for a multidomain scenario. Both templates used the same previously defined NSD(Figure 4.9b) but differed in the number of subnets configured in each NST. The first template was developed to test the inter-domain feature in a single domain, containing two independent subnets to serve as the tunnel endpoints. The second NST, aimed to test the mechanism in a multidomain scenario, defining only one subnet since each tunnel peer should be in distinct domains and slices. Figure 4.10 presents both topologies to help illustrate their implementations and differences.



(a) Internal interdomain NST topology

(b) Internal interdomain NST with 1 NSST topology

Figure 4.10: Topologies of both NSTs created

Below I present an example of the templates created, specifically a portion of the NST developed for the single domain deployment. As already mentioned, the template defines two

network slice subnets and the service quality the slice should provide.

```
nst:
- id: interdomain_nst
  name: interdomain_nst
  SNSSAI-identifier:
      slice-service-type: eMBB
  quality-of-service:
      id: 1
  netslice-subnet:
  -   id: interdomain-tunnel-peer
      nsd-ref: interdomain_slice_nsd
  -   id: interdomain-ns_2
      nsd-ref: interdomain_slice_nsd
...
```

4.7 Implementation Hardships

During the implementation phase of the NetOr project, there were several challenges and problems I needed to solve. Many times, when a problem appeared, the problem seemed to always fall always in the same set of culprits. I found and solved problems in four main areas, namely OSM problems, integrations between OSM and Openstack, unexpected modifications in OSM, and construction of NetOr's higher-level structures.

The OSM problems consisted of several inconsistencies and bugs from various services of the OSM platform, such as its web portal and its backend service, that ended up delaying my platform implementation. Some OSM limitations persist: the first section of any entity name cannot end with a non-alphabetic character; the platform doesn't support NSI operations, although defining and documenting them; the NSI deletion operation provided by the web portal and Python client executes a termination instead, leaving the instance terminated but not deleted; and finally, the incomplete support of NSI VLDs, which although supported by the NST data model, it is not accessible through the instantiation configurations.

Concerning the OSM and Openstack integrations, those problems consisted of miss configurations between both platforms. For example, when onboarding the new Openstack VIM, since it has specific projects and groups associated with a user, that information should be carefully detailed in the OSM, which can go unnoticed since it is not mandatory. Additionally, also related to this integration, the resources' identifiers, such as image, flavor, and networks used by the OSM entities, should correspond to those defined in the Openstack. The VNFDs, NSDs, and NSTs creations should follow those restrictions so that the service instantiation will work correctly and not encounter any VIM errors.

Another problem encountered was the unexpected shift in some aspects of the OSM platform, such as the Juju charm behavior and the VNFD structures. Those problems considerably delayed the implementation phase since these modifications were not notified or

documented, being deployed with no notification, altering dependencies and behavior. When dealing with this problem, it proved to be very challenging not having documentation since the origin of the error was not clear, given that the code and configurations developed by me were the same, meaning that it stopped working without apparent reason. Concerning the VNFD structure update, after encountering the problem, I verified if that structure changed as a sanity check, but it proved to be one cause of the problem. Concerning the Juju charm behavior modification, with the help of colleagues, we found the problem origin, which consisted of the OSM Juju engine using dependencies with different versions from those used initially. This new dependencies version was not compatible with the structure and approach used for the developed Juju charm. This unexpected shifts occurred while using a freezed version of the OSM platform.

Lastly, another area that caused and may cause problems when implementing this system is creating new high-level structures, such as Vertical Service Blueprints, Vertical Service Descriptors, and Network Slice Templates. That may cause problems since the structures and values need to follow strict conditions and guidelines so that the NetOr system correctly processes them and the service deploys with success. Those types of restrictions are common in all SoA vertical service orchestration solutions. Besides the strict structures themselves, the fields that need more attention are those using identifiers referring to entities in other platforms, such as the NFVO and VIM. Those fields are the ones that allow the creation of the service.

4.8 Summary

The work developed for the NetOr POC can be divided into four main areas:

- NetOr backend server development
- NetOr frontend Web Portal development
- NetOr system deployment
- NFVO resources development

The NetOr backend server is carefully detailed in this chapter, presenting and explaining the system's micro-service and event-driven architecture, its sub-components, and implementation choices. Special attention was given to describing the standards and reference projects I used to decide the information models and operations supported in the NetOr system. Finally, a detailed description of how the inter-domain approach was designed and what is the NetOr's paper in this scenario was presented. A novel approach encountered during the system's development phase was also studied.

Once the backend server was developed, a web portal was developed to facilitate the interaction between the vertical and the NetOr system. This portal was developed with clean and minimalist aesthetics, intuitiveness, and functionality in mind. With all components developed, the deployment strategy was also analyzed and presented, where I concluded that the best approach was to use Docker and containerize the NetOr system, attributing each micro-service to an isolated container.

Finally, the implementation hardships encountered during the entire development of the NetOr system were mentioned, being the most impactful ones the OSM problems and the OSM unexpected modifications. The first problem created some development delays since being OSM one of the most used NFVOs, I expected that the functionalities considered implemented by the platform's documentation were available, being quite challenging debugging a problem where the issue is the functionality itself. The second problem also caused some challenges and delays since, despite the version freeze performed, the platform behavior shifted unexpectedly, causing errors without apparent reason.

CHAPTER 5

Results

This chapter presents the testing environment, tests conducted and results obtained from validating the NetOr system. Each set of results gathered are complemented with a discussion and justification for the behavior of those values. Finally, the inter-domain is also presented as a result due to it being integrated and featured in different projects and scientific works.

5.1 Testing Environment

To test and validate the entire NetOr system by performing different tests, I created a testing environment. Since one of the main features these tests validate is the inter-domain mechanism, there was an obvious need for having at least two domains. Each domain consists of an NFVO, such as OSM, that communicates with a VIM, such as OpenStack.

For the inter-domain environment, I created two independent OSMs, one for each domain, and each one deployed in a VM, running *Ubuntu 18.04*, with 12 Gb of RAM, 4 VCPUs, 150 Gb of storage. The first OSM integrates with a fully-fledged OpenStack instance managed by IT-Aveiro, where the project defined had the limitations of 30 VM instances, 60 VCPUs, 70 Gb of RAM, 1 Tb of storage, and 10 networks. The second OSM integrates with a DevStack, a version of OpenStack focused on allowing the quick creation of development infrastructures, managed by a colleague in IT-Aveiro. The project assigned had the limitations of 10 VM instances, 20 VCPUs, 50 Gb of RAM, 1 Tb of storage, and 100 networks.

Additionally, I used a dedicated VM running *Ubuntu 18.04* with 8 Gb of RAM, 4 VCPUs, and 32 Gb of storage to deploy the NetOr system. By doing this, only the NetOr system operated in that machine, which removed any interferences of other services that may consume large amounts of resources.

Lastly, when performing comparison tests between the NetOr system and the selected SoA project, the 5GR-VS in this case, both platforms were deployed in the same machine to remove any hardware or OS-induced variance. That machine had 16 Gb of RAM, 8 CPUs, 100Gb of storage, and ran *Ubuntu 20.04*.

5.2 Tests Performed

Since NetOr is a complete platform composed of a micro-service-oriented backend service and a frontend Web Portal UI, to validate the entire system, I needed to perform several types of tests to assert the quality of all the components. Those tests were: unit tests for validating functions and algorithms of the NetOr backend micro-service system; integration tests to validate the communications and interactions between each micro-service; performance tests to validate the efficiency of the final platform; load tests to assert which is the load the NetOr system can successfully handle by simulating a real-life production scenario; and finally, usability tests to assert if the developed web portal is intuitive and efficient enough for the intended purpose.

Starting by detailing the unit tests designed and implemented, below I present a list of the tests performed for each component of the NetOr system. Those may range from testing the REST API operations provided by each micro-service to crucial algorithms and mechanisms. All the implemented tests were successful when executed.

VSB/VSD/NST Catalogue Service (Serializer):

- Create a VSD parameter value range with invalid id
- Create VSD to NSD Translation Rule with invalid input, ids or versions
- Onboard VNF Package with invalid name, version, provider, checksum or package path
- Onboard VSB without data
- Create VSB parameter with invalid id
- Create VSB Component with invalid id or VSB id
- Create VSB Forwarding Graph with invalid component or endpoint
- Create VSB endpoint without id
- Create VSB with invalid version, name or slice service category
- Onboard VSD with invalid name, version or VSB id

Tenant management Service (API):

- API Request Without Authorization
- Get all Groups
- Get Group By Id
- Get all Groups After Post
- Create New Group
- Delete Existing Group
- Get all Tenants
- Get Tenant By Id
- Get all Tenants After Post
- Create New Tenant
- Delete Existing Tenant

Domain Management Service (API):

- API Request Without Authorization
- Get all Domains
- Get Domain By Id
- Get all Domains After Post
- Create New Domain
- Delete Existing Domain

VSI/NSI Coordinator Service (API):

- API Request Without Authorization
- Get all VSIs
- Get Domain By Id
- Get all VSIs After Post
- Create New VSI
- Delete Existing VSI

VSI/NSI LCM Management Service (Mechanisms):

- Propagates error if some error occurred in the intermediate steps
- Processes correctly if all information is aligned
- Processes correctly the inter-domain mechanism when needed

Placement Service (Mechanisms):

- Propagates error if some error occurred in the intermediate steps
- Processes correctly if all information is aligned (NS subnets)
- Processes correctly if all information is aligned (NSI subnets)

Concerning integration tests, the adopted approach was to perform E2E tests to validate all integrations at once, validating both the portal and the entire NetOr backend server integrations. The list of the tests conducted for that purpose is presented below. All test were successful when executed, proving that the NetOr interfaces are aligned between components.

- Get all Default Groups
- Create New Group
- Delete Existing Group
- Get all Default Tenants
- Create New Tenant
- Delete Existing Tenant
- Get Preloaded Domain
- Create New Domain
- Delete Existing Domain
- Create New VSB
- Create New VSD
- Create New VSI
- Delete Existing VSI
- Delete Existing VSD
- Delete Existing VSB
- Delete Existing NST

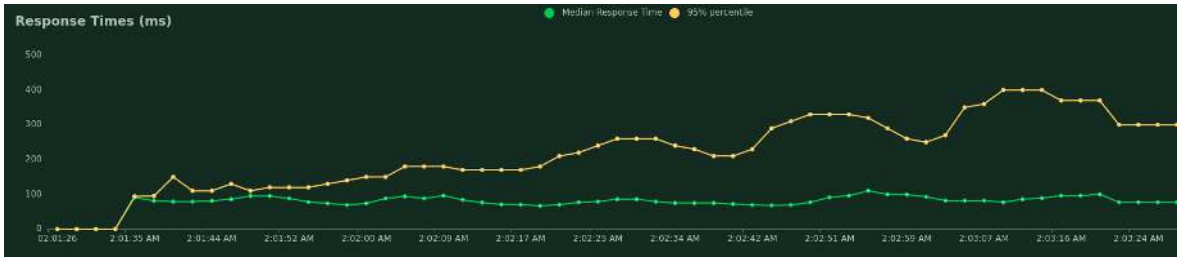
5.2.1 Load Tests

The load tests performed over the NetOr platform needed to follow some limitations since the test environment goal simulated the necessary domain, meaning that the testing infrastructure had restrictions and fewer resources than desirable. That means that the load tests needed to be on a smaller scale so that the infrastructure could safely handle the requests made and users created. For that purpose, using the Locust platform, I developed these tests to validate the platform's capability of handling considerable amounts of users and requests. The tests performed focus specifically on simulating the creation of various users, followed by each one requesting the instantiation of a previously defined Vertical Service, which simulates a real-life scenario if the platform became the go-to vertical service orchestration system.

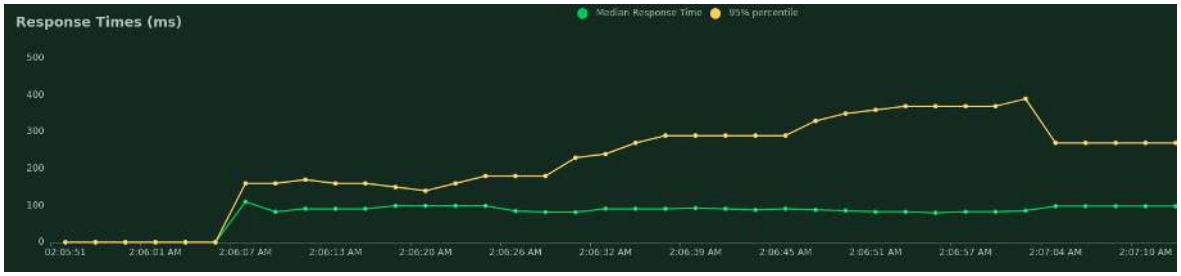
Due to the mentioned limitations, I decided to test the NetOr system independently, skipping the stage of instantiating the vertical service in the various domains. This test is still quite valuable since it allowed me to analyze the NetOr system itself and understand its current limitations. Figure 5.1 presents the results of all load tests, all aimed at creating 100 users in total, differing only in the rate they spawn. That means that during the test execution, the system experiences an increase in load since all previously created users and management entities are active until the end of the test. I performed three different load tests, spawning users at 1, 2, and 3 users per second, corresponding to 3, 6, and 8 requests per second, respectively. I executed all three scenarios over a simple instantiation of the NetOr system, meaning there was no component replication.

After analyzing the graphs presented in Figure 5.1, specifically the Sub-figure 5.1a and Sub-figure 5.1b, I conclude that the system handles quite well a slow creation of new users, increasing the response time but keeping it in a reasonable range, never surpassing the 400 milliseconds threshold. On the other hand, if the number of users per second increases, as shown by Sub-figure 5.1c, the simple instantiation of the NetOr system has problems keeping up with the load generated, increasing the response time in the final third of user creation up to 1,5 seconds. This behavior is most likely due to resource limitations, and scaling the bottleneck component could solve the problem.

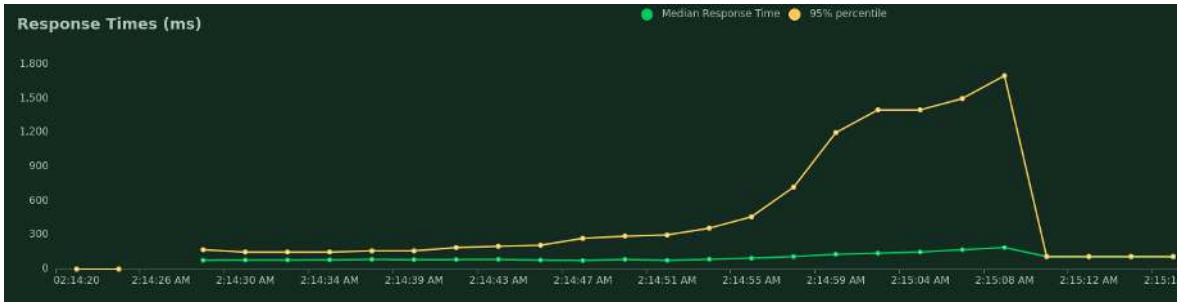
Note that the scenarios simulated by these tests are a conscious exaggeration of real-life scenarios. Even the one user/second test is considerably more demanding than a real-life scenario since systems like the NetOr are typically closely managed and provided to pre-defined entities. That means that the creation of 100 consecutive users is quite difficult to happen in reality. Similarly, having 100 vertical services instantiated in a short period would also be very rare since it isn't normal for verticals to request that volume of services. Verticals' expected behavior is creating services sporadically to provide a new service or extend existing ones to their clients.



(a) Creation of 100 users with a spawn rare of 1 user/second



(b) Creation of 100 users with a spawn rare of 2 user/second



(c) Creation of 100 users with a spawn rare of 3 user/second

Figure 5.1: Load tests results

5.2.2 Performance Values

In terms of performance tests executed to assess and validate the system behavior in terms of delays, I conducted system-focused and E2E tests over the NetOr system. Both tests were compared with the same tests executed with the 5GR-VS, currently the most mature vertical service orchestration solution, assessing if the values generated are comparable.

The first test focused on the delay added by the orchestration system to the service instantiation and termination processes, calculating the delay until the NFVO was contacted. These delays correspond to the *System Instantiation Delay* and *System Termination Delay* stages presented in Figure 5.2. The other performance test focused on validating the entire E2E flow of information, starting at the NetOr backend server, followed by the NFVO, and finally the VIM. With this platform stack defined, it was possible to create an E2E vertical service, testing both creation and termination success and the time intervals needed to perform those actions. These E2E delays correspond to the *E2E Instantiation* and *E2E Termination* stages presented in Figure 5.2.

The high-level descriptors and blueprints used in these tests are presented in Appendix B.

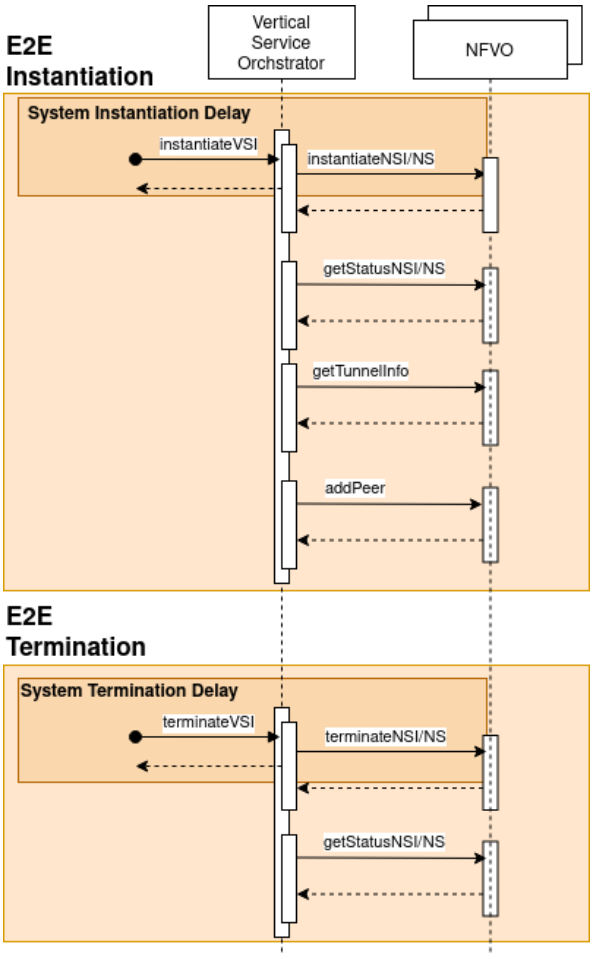


Figure 5.2: Stages tested

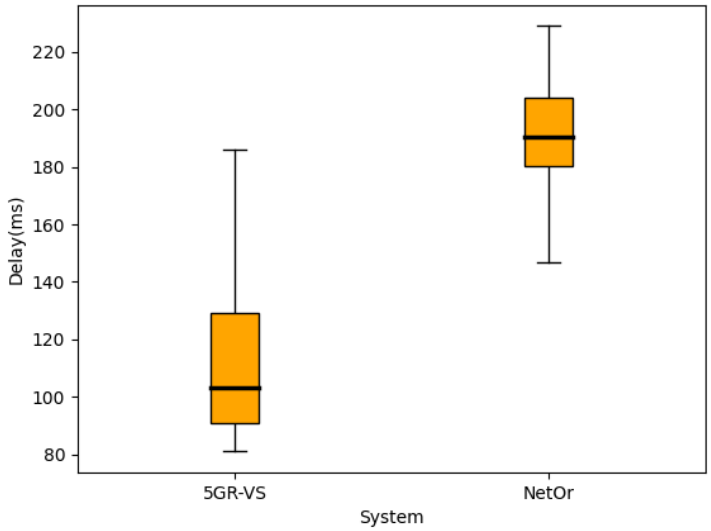


Figure 5.3: 5GR-VS and NetOr system's service instantiation delay

System	Max	Min	Avg	StDev
5GR-VS	234 ms	81 ms	113,86 ms	29,20 ms
NetOr	345 ms	135 ms	190.93 ms	24.82 ms

Table 5.1: 5GR-VS and NetOr System's service instantiation delay

Orchestrator System Delays

The first comparison test focused on the delay added by the NetOr and 5GR-VS systems to the vertical service instantiation process. To obtain that delay, I calculated the time delta between the exact moment before making the instantiation request and the precise moment before each system forwarding it to the respective underlying NFVO.

Both Figure 5.3 and Table 5.1 present the same data. By analyzing them, it is clear that the NetOr system generates higher delays than VS, but the difference between them is not that significant, being on average lower than 80 milliseconds. This difference is mainly due to the systems' implementation styles since the NetOr micro-services oriented architecture demand a higher number of communications between components, causing an increase in the process's overall delay compared to the VS monolithic approach. Nevertheless, both systems' delays are in the same order of magnitude, and in this test, the NetOr had less value deviation.

System	Max	Min	Avg	StDev
5GR-VS	108 ms	19 ms	45,90 ms	17,20 ms
NetOr	187 ms	59 ms	101,24 ms	20,90 ms

Table 5.2: 5GR-VS and NetOr System's service termination delay

The second comparison test was very similar to the first one, but it compared the delay of the system when terminating a vertical service. As processed in the first comparison test, I also obtained the time interval by calculating the time delta between the moment before making the termination request and the moment before each system forwarding those actions

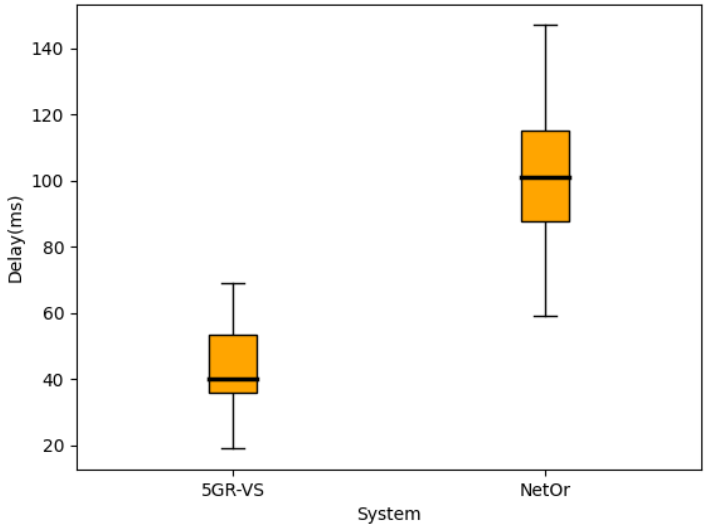


Figure 5.4: 5GR-VS and NetOr System's service termination delay

to the respective underlying NFVO.

Once again, both Figure 5.4 and Table 5.2 present the same data. Following the same behavior shown in the first comparison test, the NetOr system generates higher delays than the VS, differing on average less than 40 milliseconds. The reason for this difference is once again the fact that a micro-service-oriented system needs more network communications to exchange data between the different components of the system, which in the end increases the overall process time.

Note that the delay values obtained from this termination operation are considerably lower than those obtained from the instantiation operation. This behavior was expected since the instantiation process is much more complex than the termination of a vertical service. During instantiation, both systems need to translate data models, calculate placement restrictions, create several management entities, and trigger the correct actions in the underlying NFVOs. The termination operation requires only the teardown of those management entities and signaling the termination of the service to the NFVOs.

E2E Management Delays

System	**Max**	**Min**	**Avg**	**StDev**
Instantiation	409346 ms	279582 ms	348858,97 ms	29334,11 ms
Termination	109319 ms	38687 ms	74971,93 ms	16588,79 ms

Table 5.3: E2E interdomain service instantiation and termination delay

Concerning the E2E tests performed with the NetOr system, the first one focused on validating and gathering the time needed to instantiate and terminate a fully-fledged inter-domain service. The final service was composed of an NSI(managed of the NetOr system) composed by two distinct NSSIs, one for each domain. Each of those NSSIs contained an NSI with the same architecture as presented in Sub-figure 4.10b.

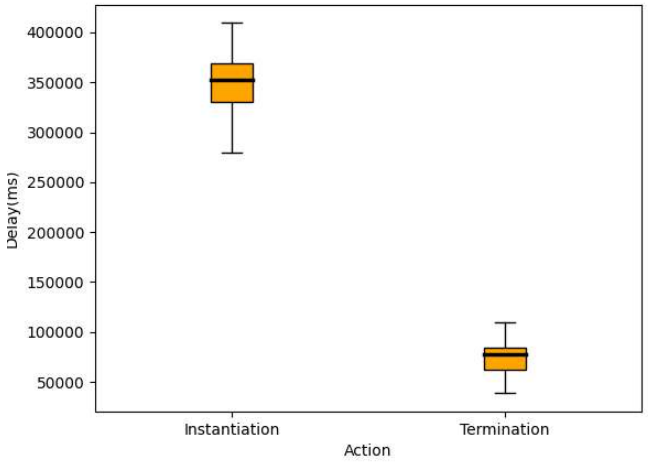


Figure 5.5: E2E interdomain service instantiation and termination delay

When analyzing both Figure 5.5 and Table 5.3, which represent the same data, I can conclude that, as expected, the instantiation time intervals are considerably higher than the termination time intervals. As presented in Subsection 5.2.2, this phenomenon happens because the instantiation process is more complex than the termination operation. The system needs to spend more time and resources to instantiate a vertical service.

Another relevant conclusion from this test is that the time intervals obtained for these E2E operations are dependent on the infrastructure used to support the creation of those services and the necessary VMs. As presented in Subsection 5.2.2, it is safe to assume that the delay added by the NetOr system to the instantiation process is approximately 200 milliseconds. With that in mind, most of the time spent instantiating the network service is the NFVO and the VIM creating the necessary resources. For that reason, the time delays of approximately 5,8 minutes for the instantiation and around 1,2 minutes for the termination operations can be lower if the infrastructure allows.

Concerning this E2E inter-domain service management delays study, I helped in the development of [32], which focuses on this topic, but using the 5GR-VS orchestrator. Given that one of the tests conducted on that article considered the same inter-domain service as the one in this POC, and it was validated in the same testing environment as the one used for this POC, this enables the direct comparison between the E2E management delays obtained in the NetOr and the 5GR-VS orchestrators. According to [32], this simple inter-domain connectivity service when orchestrated by 5GR-VS took on average 9 minutes to instantiate and 46 seconds to terminate the entire service. When compared to the NetOr values, the instantiation delays in the 5GR-VS is considerably higher, differing in more than 3 minutes. The termination delays originated by the 5GR-VS is lower than the NetOr ones, differing close to 30 seconds. The difference between the instantiation delays can be easily justified by the systems' implementation approaches. The 5GR-VS is a monolithic platform that sequentially manages its services, meaning that at any given time, the system can only process

one operation of a given service component. On the other hand, the NetOr system, aided by its micro-service-oriented architecture, can manage various service components and various services at the same time. This parallelization enables the NetOr platform to save precious minutes by processing the independent service subnets at the same time. The reason for the 5GR-VS generating lower E2E termination delays is harder to understand, but I concluded that, given that the overall process is quite simple and quick, performance variations in the underlying infrastructure can affect the E2E delays of this phase.

A conclusion to retain from this test is the meaning of the values obtained. Although 5,8 and 1,2 minutes seem high delays to instantiate and terminate a service, it is crucial to understand that these times correspond to the instantiation of an E2E NSI that manages other two NSIs as its NSSIs, which are dynamically interconnected in domains without prior knowledge of each other. These delays are significantly lower than those currently possible since permissions and domain communication channels tend to be pre-defined, which may take days or weeks.

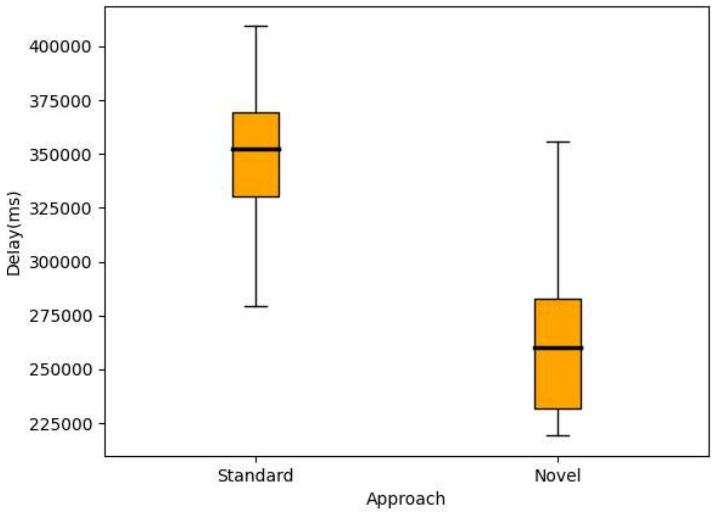


Figure 5.6: Standard and Novel interdomain approaches instantiation delay

System	Max	Min	Avg	StDev
Standard Approach	409346 ms	279582 ms	348858,97 ms	29334,11 ms
Novel Approach	355886 ms	219143 ms	264136,83 ms	37086,75 ms

Table 5.4: Standard and Novel interdomain approaches instantiation delay

As presented in Subsection 4.3, although not the default inter-domain mechanism used by the platform, the NetOr system still supports it. For that reason, and because this approach can have some interesting capabilities and behavior, I decided to test it and gather the time intervals needed to instantiate an interdomain service.

Both Figure 5.6 and Table 5.4 present the same data. As clearly depicted in Figure 5.6, this novel approach achieves significantly lower times when instantiating an inter-domain service. The difference, on average, between approaches can reach 84722 milliseconds, approximately 1,4 minutes. The expected behavior of these types of systems easily explains this difference.

As proposed by existing standards and implemented by available vertical service orchestration solutions, the communications in the NFV stack should never skip an abstraction level, meaning that a given layer only interacts with those immediately next to it. Consequently, if a higher-level layer wants to update the status of lower-level ones, it needs to request the new information, originating the need to create pooling agents that continually update the information of lower layers. Depending on the implementation, the time interval between each update can vary.

Since the inter-domain mechanism needs to dynamically establish the connection between all domains through VNF-level Day-2 operations, the NetOr system needs to continuously update the service's information until it is configured and running. From that moment on, it is possible to perform runtime operations over the vertical service. The novel approach presented proposes to solve this problem with an event-driven solution, meaning that the tunnel peer VNF, when fully configured, sends a message to an exterior pre-defined agent that will handle the interdomain mechanism, which in this case is the NetOr. Since the NetOr implementation defines that a new vertical service information update should occur every minute, that is the reason for the delay difference between approaches being close to 1 minute (1.4 minutes).

A final performance test focused on validating and checking the performance of the tunnel itself, assuring the VPN tunnel configuration didn't deteriorate the connection quality. Since these tests are being performed in a controlled infrastructure, I could define the expected connection bandwidth, which in this case was limited by 1Gb/s. After instantiating an inter-domain vertical service and using the iPerf[1] tool, I tested the bandwidth allowed by the defined tunnel. The result was 824 Mbit/second, which is approximately 1Gb/Second. This result is very promising since it indicates that these inter-domain connectivity mechanism doesn't alter in a significant manner the connection established.

5.2.3 Portal Specific Tests

I also planned the execution of portal-specific tests to assert its functionality, usability, and user experience quality. In addition to the already mentioned integration tests, which validate portal integrations and the functionalities it provides, I also considered it necessary to perform tests only to the portal, namely usability and user experience tests. To do that, the plan was to use a heuristic-based test to evaluate the portal, using the most well-known and used heuristic evaluation set, which is Nielsen's heuristics. To perform this type of test, several evaluators should analyze the UI based on the heuristics proposed by Nielsen and associate a severity level to each of them. In Nielsen's heuristic test, the severity scale goes from 0 to 4, where 0 means that there is no usability problem and four means that the interface has a catastrophic failure.

Unfortunately, to obtain significant results from this type of test, it needs a considerable number of evaluators, such as 10, to successfully identify interface problems and their severity. Since the NetOr system handles complex concepts and ideas, it requires additional knowledge

[1] https://iperf.fr/

about those concepts and entities to perform any action in the platform. For that reason, gathering ten distinct evaluators with such knowledge proved to be quite challenging. For that reason, I skipped this test due to those restrictions, but an example with only two evaluators is present in Table 5.5.

Heuristic	**Evaluator 1**	**Evaluator 2**
Visibility of system status	0	0
Match between system and the real world	0	1
User control and freedom	0	0
Consistency and standards	0	2
Error prevention	1	0
Recognition rather than recall	0	2
Flexibility and efficiency of use	2	0
Aesthetic and minimalist design	0	0
Help users recognize, diagnose, and recover from errors	0	2
Help and documentation	1	1

Table 5.5: Nielsen Heuristic Evaluation

5.3 Multi-domain Innovation

This mechanism is a powerful feature, possibly composing one of the next steps in the NFV and operators world, which is very sought after.

As a consequence, a version of this inter-domain mechanism is expected to be integrated and tested in 5Growth, a European project with the most mature state-of-the-art vertical service orchestration solution. The 5Growth European project interacts with different vertical industries to assert and test the innovations developed for it, namely Efacec[2] Energia, Efacec Transportes, Comau[3], and Innovalia[4]. Each of those industries defined one or more use cases specific to their objectives and day-to-day operations, scenarios that the project will test in several pilots. The proposal is to integrate the inter-domain innovation into at least one of the four pilots. Those efforts are being made mainly by a team in Nextworks[5] and a team in IT Aveiro[6], which I integrate.

The integration of that mechanism in the 5Growth project was possible through the combined work of multiple organizations, namely IT Aveiro, Nextworks, Altice Labs[7], and Efacec. The pilot this inter-domain mechanism will integrate is composed of three domains, one with the portal and services necessary for the Vertical, another simulating the premises and infrastructure of a provider orchestrated with OSM, and the last one mimicking the premises and infrastructure of a provider choreographed with SONATA[8]. All those domains are expected to be connected through fully-fledged 5G networks.

[2]https://www.efacec.pt/
[3]https://www.comau.com
[4]https://innovalia.org
[5]https://www.nextworks.it/
[6]https://www.it.pt/
[7]https://www.alticelabs.com
[8]https://www.sonata-nfv.eu/

Integrating the IT Aveiro's 5Growth Team, I was involved in the implementation and integration of that inter-domain innovation in the 5Growth project, contributing to several deliverables, such as the D2.1 [28], D2.3 [33], D4.2 [34] and D5.4 [35]. Additionally, a version of that innovation was already presented to the 5Growth's European committee during a mid-term project demonstration, being accepted. A more detailed description of the work and integrations needed for that pilot test, as well as the inner working of the inter-domain innovation itself is presented in [36], where I also contributed.

Finally, the inter-domain mechanism was also featured in [32], a work focused on the dynamic network slicing orchestration possibilities for verticals when integrating the inter-domain mechanism into the 5Growth project. The article focuses on the orchestration delays generated when instantiating, modifying, and terminating an inter-domain vertical service.

5.4 Summary

This chapter presents all tests conducted over the NetOr system and the results obtained from them. Some of the tests were unit, functional, and integration tests that assured the correct functioning of the crucial APIs and algorithms developed in NetOr. Portal tests were also mentioned and planned to be performed. Unfortunately, due to the lack of qualified evaluators, these tests were only explained and exemplified to enable their execution in the future.

Performance-related tests were also executed, such as load tests that identified what is the maximum load the NetOr system supports, and if it handles common vertical service orchestration loads. Based on the performance tests and the delays registered when managing an inter-domain service with the NetOr system, I also compared those values with the ones obtained from orchestrating the same service with the 5GR-VS, the most mature SoA vertical service orchestration system. The conclusions from these comparisons proved that the NetOr system is a competitive platform to other SoA systems.

Finally, the inter-domain mechanism was also considered as a result of this POC development, since this functionality is being adopted and featured in a European project and different scientific papers.

CHAPTER 6

Conclusions

As seen by the results when comparing the NetOr's performance values to those obtained from 5GR-VS, there are positive and negative aspects. By following a micro-service-oriented architecture, the NetOr generates higher delays in small and quick operations due to the need for higher network communications between components. On the other hand, comparing the performance of more complex processes that can take a substantial time, the NetOr performs better than the 5GR-VS by taking advantage of its parallelization capabilities.

Additionally, the NetOr platform already provides options and mechanisms that the 5GR-VS does not. The NetOr system has two main improvements over the VS: the support of instantiation configuration parameters, which are propagated down the NFV stack to allow dynamic configuration of network slices and services; the support of service configuration and runtime operations, also known as Day-1 and Day-2 operations. By defining the actions supported by the services, their parameters, and default values in the VSBs, the NetOr system provides a way of executing runtime operations over the running services.

After developing and testing the NetOr system, I conclude that, although some of the performance results show that this new platform is not the best compared with an already existing solution, such as the 5GR-VS, I think that the benefits obtained from the implementation style compensate those values. Although with higher system delays, when comparing the NetOr delays to 5GR-VS's ones, the difference encountered is almost irrelevant, being always less than 100 milliseconds, which means that both platforms generate delays in the same magnitude order. I already expected that delay difference because although a micro-service-oriented system allows parallel processing that can decrease the overall process time, it needs more communications between components through the network, which increases the overall time of the procedure. Nevertheless, the usability, scalability, and maintainability gained by the new system outweigh the delay gained.

The development of this system was challenging since it demanded the integration of many diverse platforms, their data models, and operations. Nevertheless, the final NetOr platform is a promising solution that successfully implements the defined use cases.

Finally, the NetOr system was already presented in the Rede Temática de Comunicações Móveis(RTCM) Seminary[1].

6.1 Future work

The NetOr is an extensive and ambitious project that competes with very complex platforms, most of them several years-long projects developed by senior developers of the area, such as 5GR-VS.

With that in mind, I had to prioritize each component of the NetOr system and implement only those needed to fulfill the defined use-cases. Therefore, due to time restraints, some functionalities and elements were not paramount for the main workflows this NetOr POC wanted to demonstrate, so they were not totally or partially implemented. The components and functionalities that future NetOr contributors should implement are:

- Monitoring Service
- SLA Management
- Update/Modification of Services
- NSI Management Improvement
- Tests with a different technology domain

The Monitoring Service should be a component to consider in the future since it adds many advantages and possibilities to the system. By simply deploying an Elastic Search, Logstash and Kibana (ELK)[2] stack and gathering the logs of all micro-services, it becomes significantly easier to maintain and debug the platform by having a centralized logging agent. In this approach, it is also possible to define alarms, which facilitate identifying a given problem and even act upon its occurrence. Another option is to use technologies with richer features, such as Telegraf, InfluxDB, Chronograf and Kapacitor (TICK) Stack[3] or Prometheus [4], which can process more complex data, defining specialized alarms, and automate actions.

The SLA Management is another aspect that was not crucial for this POC but is very important in systems like the NetOr. The SLA defines the interaction guidelines between a client and a provider, specifically between a vertical and the service/network provider in this scenario. That agreement establishes performance values and resources limitations for both sides of the interaction, assuring that once defined, the provider will follow those values as close as possible, enabling the vertical to achieve its KPIs and keep its reputation.

The update/modification of services was one of the features partially implemented. The NetOr platform already has robust support of operations over the instantiated VSIs, specifically Day-2 operations, which can modify the service. Another approach to update/modify a service is to scale it. From my SoA analysis, even the currently available solutions do not fully support this functionality, which is another aspect to improve. I think that service scaling

[1] https://rtcm.inesctec.pt
[2] https://www.elastic.co/what-is/elk-stack
[3] https://www.influxdata.com/blog/introduction-to-influxdatas-influxdb-and-tick-stack/
[4] https://prometheus.io/docs/introduction/overview/

should be considered in the future for the NetOr system since it is a powerful functionality necessary for many real use-cases. An example is what happens many times with most medium-to-large-sized software systems, which due to the load those systems undergo, they need to replicate some components, scaling up or down as necessary.

Although the NetOr platform already considers and manages the NSIs associated with the several vertical services instantiated, improving their management mechanism can open options and functionalities that can benefit the NetOr system. Having a better NSI management allows the usage of orchestrators that lack that slicing management and increases the NetOr's integration possibilities.

Finally, even though the NetOr system is aligned with the correct standards and implemented with the necessity of integration with different NFVO technologies, tests over that interoperability functionality were not performed. Drivers for each new technology need to exist to enable communication with each new orchestrator. As future work, I propose for those tests to be executed, validating the mechanism and asserting which technologies the data model already supports.

References

[1] *VERTICAL | meaning in the Cambridge English Dictionary*. [Online]. Available: https://dictionary.cambridge.org/dictionary/english/vertical (visited on 11/17/2020).

[2] 5GPPP, "Empowering Vertical Industries through 5G Networks - Current Status and Future Trends," pp. 1–108, 2020. DOI: 10.5281/zenodo.3698113. [Online]. Available: http://doi.org/10.5281/zenodo.3698113.

[3] 5G_PPP, "View on 5G Architecture," *Version 3.0, February 2020*, no. February, pp. 21–470, 2020. DOI: 10.5281/zenodo.3265031. [Online]. Available: https://5g-ppp.eu/wp-content/uploads/2019/07/5G-PPP-5G-Architecture-White-Paper%7B%5C_%7Dv3.0%7B%5C_%7DPublicConsultation.pdf.

[4] ETSI, "GS NFV 002 - V1.2.1 - Network Functions Virtualisation (NFV); Architectural Framework," Tech. Rep., 2014, pp. 1–21. [Online]. Available: https://www.etsi.org/deliver/etsi_gs/NFV/001_099/002/01.02.01_60/gs_NFV002v010201p.pdf.

[5] ——, "GS NFV-MAN 001 - V1.1.1 - Network Functions Virtualisation (NFV); Management and Orchestration | Enhanced Reader," Tech. Rep., 2014. [Online]. Available: https://www.etsi.org/deliver/etsi%7B%5C_%7Dgs/nfv-man/001%7B%5C_%7D099/001/01.01.01%7B%5C_%7D60/gs%7B%5C_%7Dnfv-man001v010101p.pdf.

[6] 3GPP, "3GPP TR 28.801 - v15.1.0 - Study on management and orchestration of network slicing for next generation network," Tech. Rep., 2018. [Online]. Available: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3091.

[7] A. Cardenas and D. Fernandez, "Network Slice Lifecycle Management Model for NFV-based 5G Virtual Mobile Network Operators," *2020 IEEE Conference on Network Function Virtualization and Software Defined Networks, NFV-SDN 2020 - Proceedings*, pp. 120–125, Nov. 2020. DOI: 10.1109/NFV-SDN50289.2020.9289883.

[8] 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. DOI: 10.1109/MCOM.2017.1600935.

[9] 3GPP, "Management and orchestration; Provisioning (3GPP TS 28.531 version 15.0.0 Release 15)," Tech. Rep., 2018. [Online]. Available: https://www.etsi.org/deliver/etsi_ts/128500_128599/128531/15.00.00_60/ts_128531v150000p.pdf.

[10] ——, "Management and orchestration; Concepts, use cases and requirements (3GPP TS 28.530 version 15.0.0 Release 15)," Tech. Rep., 2018. [Online]. Available: https://www.etsi.org/deliver/etsi_ts/128500_128599/128530/15.00.00_60/ts_128530v150000p.pdf.

[11] ——, "Management and orchestration; 5G Network Resource Model (NRM); Stage 2 and stage 3 (3GPP TS 28.541 version 15.0.1 Release 15)," Tech. Rep., 2018. [Online]. Available: https://www.etsi.org/deliver/etsi_TS/128500_128599/128541/15.00.01_60/ts_128541v150001p.pdf.

[12] ETSI, "GR NFV-EVE 012 - V3.1.1 - Network Functions Virtualisation (NFV) Release 3; Evolution and Ecosystem; Report on Network Slicing Support with ETSI NFV Architecture Framework," Tech. Rep., 2017. [Online]. Available: https://www.etsi.org/deliver/etsi_gr/NFV-EVE/001_099/012/03.01.01_60/gr_NFV-EVE012v030101p.pdf.

[13] ——, "GS NFV-SOL 005 - V2.6.1 - Network Functions Virtualisation (NFV) Release 2; Protocols and Data Models; RESTful protocols specification for the Os-Ma-nfvo Reference Point," Tech. Rep., 2019.

[Online]. Available: https://www.etsi.org/deliver/etsi_gs/NFV-SOL/001_099/005/02.06.01_60/gs_nfv-sol005v020601p.pdf.

[14] ——, "GS NFV-SOL 006 - V3.3.1 - Network Functions Virtualisation (NFV) Release 3; Protocols and Data Models; NFV descriptors based on YANG Specification," Tech. Rep., 2020. [Online]. Available: https://www.etsi.org/deliver/etsi_gs/NFV-SOL/001_099/006/03.03.01_60/gs_NFV-SOL006v030301p.pdf.

[15] K. Sienkiewicz, W. Latoszek, and P. Krawiec, "Services orchestration within 5G networks-Challenges and solutions," *URSI 2018 - Baltic URSI Symposium*, pp. 265–268, Jul. 2018. DOI: 10.23919/URSI.2018.8406739.

[16] P. Trakadas, P. Karkazis, H. C. Leligou, T. Zahariadis, F. Vicens, A. Zurita, P. Alemany, T. Soenen, C. Parada, J. Bonnet, E. Fotopoulou, A. Zafeiropoulos, E. Kapassa, M. Touloupou, and D. Kyriazis, "Comparison of Management and Orchestration Solutions for the 5G Era," *Journal of Sensor and Actuator Networks 2020, Vol. 9, Page 4*, vol. 9, no. 1, p. 4, Jan. 2020. DOI: 10.3390/JSAN9010004. [Online]. Available: https://www.mdpi.com/2224-2708/9/1/4/htm%20https://www.mdpi.com/2224-2708/9/1/4.

[17] F. Meneses, M. Fernandes, D. Corujo, and R. L. Aguiar, "SliMANO: An Expandable Framework for the Management and Orchestration of End-to-end Network Slices," *Proceeding of the 2019 IEEE 8th International Conference on Cloud Networking, CloudNet 2019*, Nov. 2019. DOI: 10.1109/CLOUDNET47604.2019.9064072.

[18] C. Parada, J. Bonnet, E. Fotopoulou, A. Zafeiropoulos, E. Kapassa, M. Touloupou, D. Kyriazis, R. Vilalta, R. Munoz, R. Casellas, R. Martinez, and G. Xilouris, "5Gtango: A Beyond-Mano Service Platform," *2018 European Conference on Networks and Communications, EuCNC 2018*, pp. 26–30, Aug. 2018. DOI: 10.1109/EUCNC.2018.8443232.

[19] A. Devlic, A. Hamidian, D. Liang, M. Eriksson, A. Consoli, and J. Lundstedt, "NESMO: Network slicing management and orchestration framework," *2017 IEEE International Conference on Communications Workshops, ICC Workshops 2017*, pp. 1202–1208, Jun. 2017. DOI: 10.1109/ICCW.2017.7962822.

[20] *Project Objectives – 5GinFIRE*. [Online]. Available: https://5ginfire.eu/project-objectives/ (visited on 12/28/2020).

[21] D. Gomes *et al.*, "5GINFIRE: Experimental Infrastructure Architecture and 5G Automotive Use Case," Tech. Rep. DOI: 10.18153/5IF-732497-D2_2.

[22] *Future Internet Research and Experimentation (FIRE) | Shaping Europe's digital future*. [Online]. Available: https://ec.europa.eu/digital-single-market/en/future-internet-research-and-experimentation-fire (visited on 12/28/2020).

[23] *Introduction - Openslice*. [Online]. Available: https://openslice.readthedocs.io/en/stable/ (visited on 12/28/2020).

[24] *5G-Transformer - 5G-PPP*. [Online]. Available: https://5g-ppp.eu/5g-Transformer/.

[25] C. J. Bernardos *et al.*, "5G-TRANSFORMER Refined Architecture," Tech. Rep. [Online]. Available: http://5g-transformer.eu/wp-content/uploads/2019/05/D1.3_5G-TRANSFORMER_Refined_Architecture.pdf.

[26] X. Li *et al.*, "5G-TRANSFORMER: Initial System Design," Tech. Rep. [Online]. Available: http://5g-transformer.eu/wp-content/uploads/2019/11/D1.2_5G-TRANSFORMER_Initial_System_Design.pdf.

[27] *5Growth ‹ 5G-PPP*. [Online]. Available: https://5g-ppp.eu/5growth/ (visited on 12/29/2020).

[28] A. C. Saavedra *et al.*, "5Growth D2.1: Initial Design of 5G End-to-End Service Platform," Tech. Rep. [Online]. Available: https://5growth.eu/wp-content/uploads/2019/11/D2.1-Initial_Design_of_5G_End-to-End_Service_Platform.pdf.

[29] O. Kolodiazhnyi *et al.*, "5Growth D2.2: Initial implementation of 5G End-to-End Service Platform," Tech. Rep. [Online]. Available: https://5growth.eu/wp-content/uploads/2020/05/D2.2-Initial_implementation_of_5G_End-to-End_Service_Platform.pdf.

[30] G. Bernini, P. G. Giardina, S. Spadaro, F. Agraz, A. Pages, J. Cabaca, P. Neves, K. Koutsopoulos, and A. Matencio, "Multi-domain orchestration of 5G vertical services and network slices," in *2020 IEEE International Conference on Communications Workshops, ICC Workshops 2020 - Proceedings*, Institute of Electrical and Electronics Engineers Inc., Jun. 2020, ISBN: 9781728174402. DOI: 10.1109/ICCWorkshops49005.2020.9145221.

[31] *Introduction — Open Source MANO 6.0 documentation*. [Online]. Available: https://osm.etsi.org/docs/vnf-onboarding-guidelines/00-introduction.html (visited on 07/21/2021).

[32] J. Fonseca, J. Alegria, V. A. Cunha, D. Santos, D. Gomes, P. Barraca, D. Corujo, and R. L. Aguiar, "Dynamic Inter-domain Network Slicing for Verticals in the 5Growth Project," *2021 IEEE Conference on Network Function Virtualization and Software Defined Networks, SDN-NFV 20201*, 2021, (Submitted).

[33] J. Mangues-Bafalluy *et al.*, "D2.3: Final Design and Evaluation of the innovations of the 5G End-to-End Service Platform," Tech. Rep., 2021. [Online]. Available: https://5growth.eu/wp-content/uploads/2019/06/D2.3-Final_Design_and_Evaluation_of_5G_End-to-End_Service_Platform.pdf.

[34] D. Lopez *et al.*, "D4.2: Verification methodology and tool design," Tech. Rep., 2020. [Online]. Available: https://5growth.eu/wp-content/uploads/2019/06/D4.2-Verification_methodology_and_tool_design.pdf.

[35] C. Guimarães *et al.*, "D5.4: Report on WP5 Progress and Update of CoDEP," Tech. Rep., 2021. [Online]. Available: https://5growth.eu/wp-content/uploads/2019/06/D5.4-Report_on_WP5_progress_and_update_of_CoDEP.pdf.

[36] X. I. Li, C. Guimarães, G. Landi, J. Brenes, J. Mangues-Bafalluy, P. Iovanna, J. Baranda, D. Corujo, V. Cunha, J. Alegria, A. Z. Orive, J. Ordonez-lucena, C. J. Bernardos, A. Mourad, X. Costa-perez, and J. Fonseca, "Multi-Domain Solutions for the Deployment of Private 5G Networks," *2021 IEEE Access*, vol. 4, 2021, (Accepted). DOI: 10.1109/ACCESS.2021.3100120.

Appendix A

Portal Images

Admin Pages

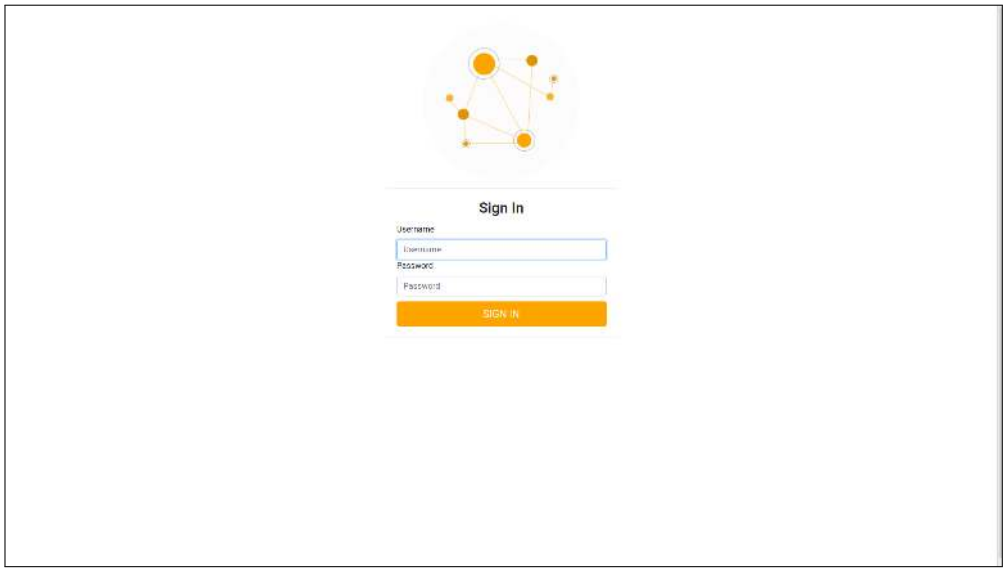


Figure 1: Admin Login Page

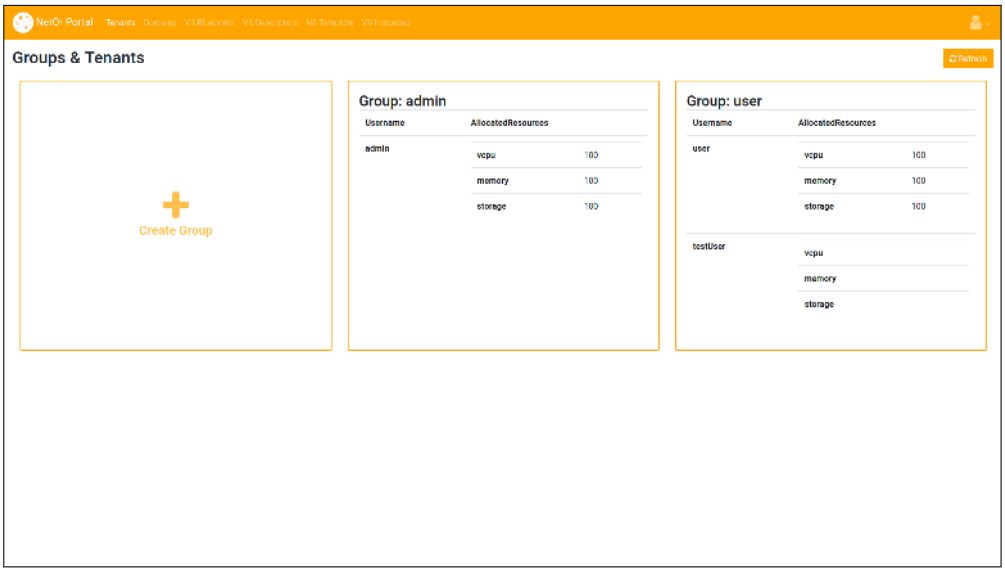


Figure 2: Admin Groups and Tenants Page

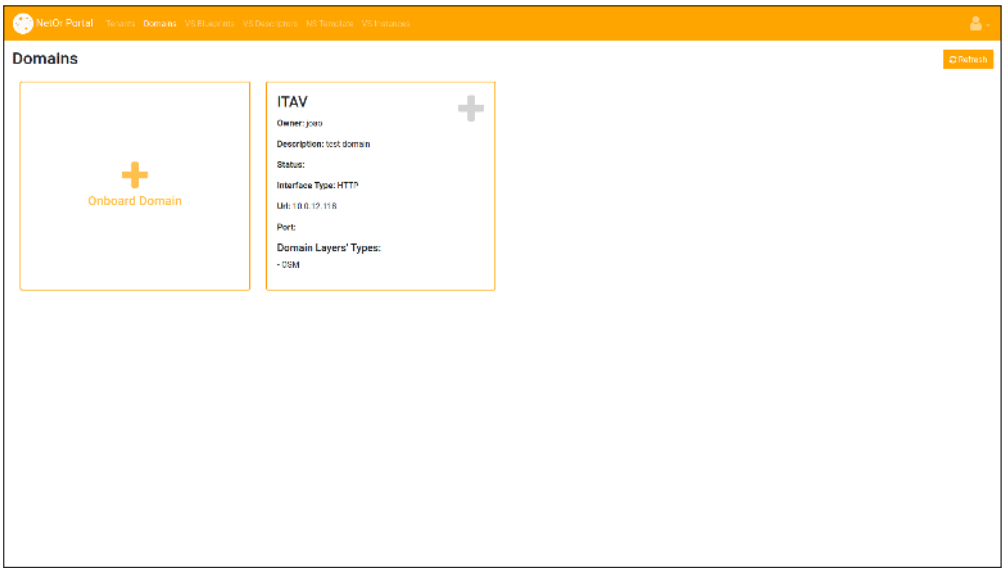


Figure 3: Admin Domains Page

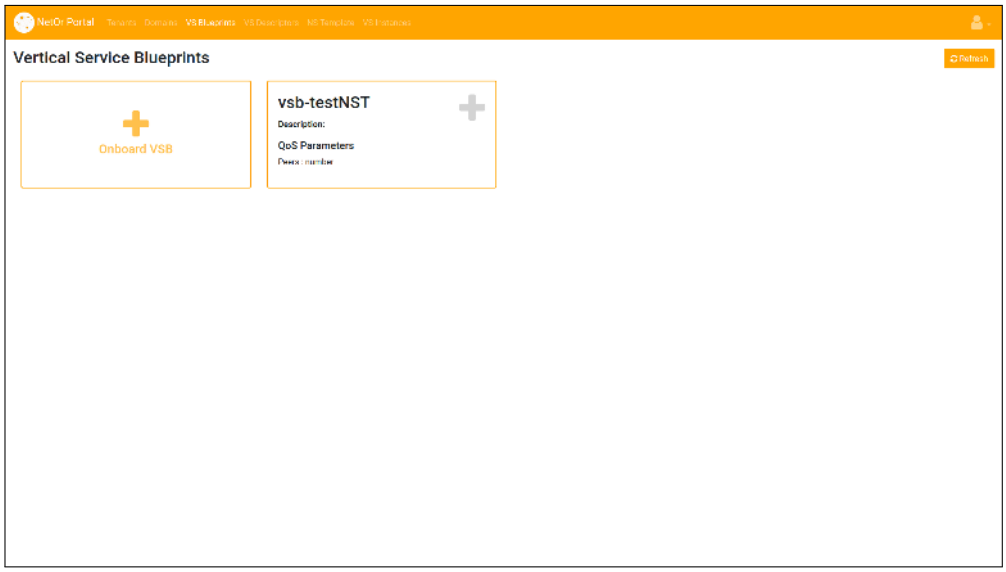


Figure 4: Admin VSBs Page

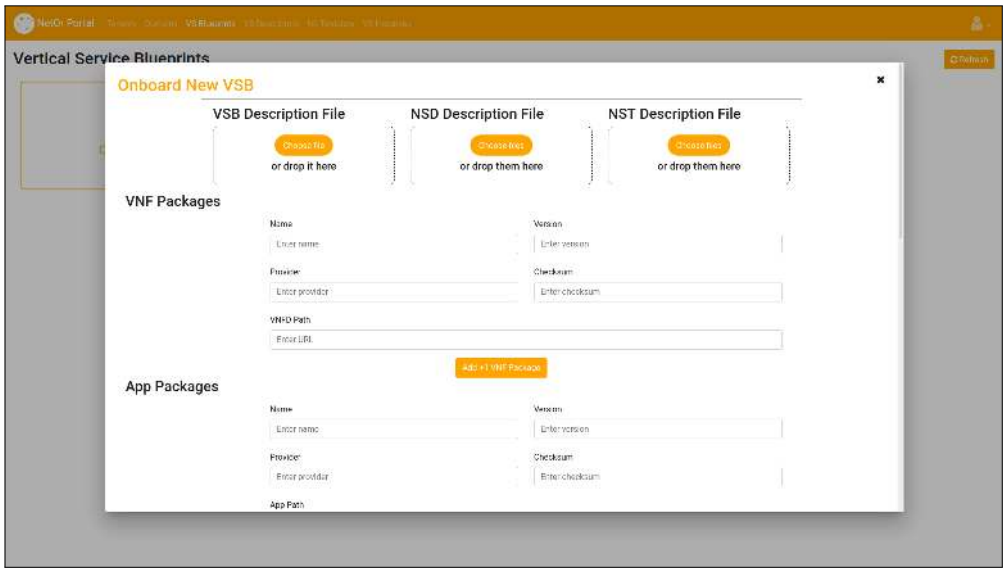


Figure 5: Admin Onboard VSB Page

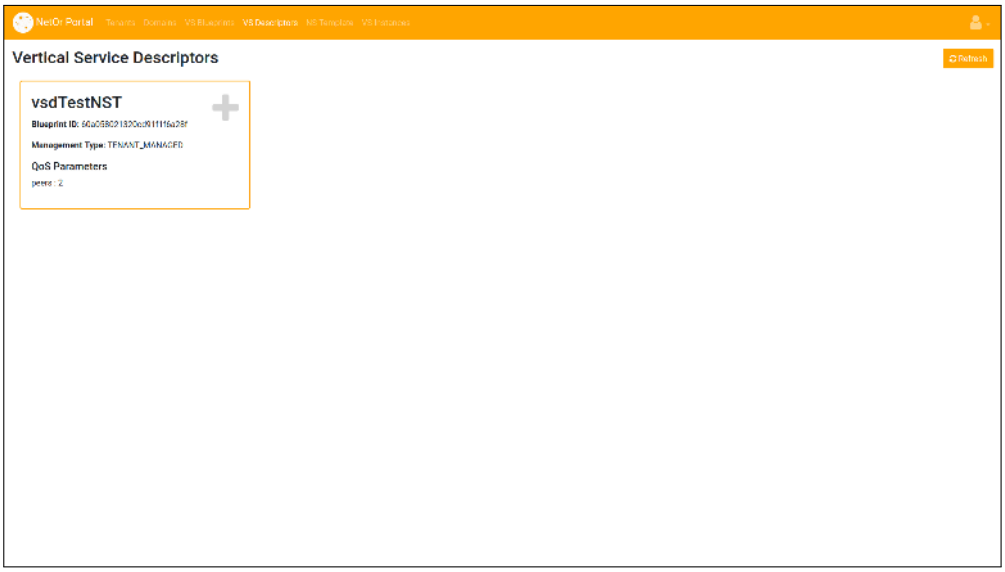


Figure 6: Admin VSDs Page

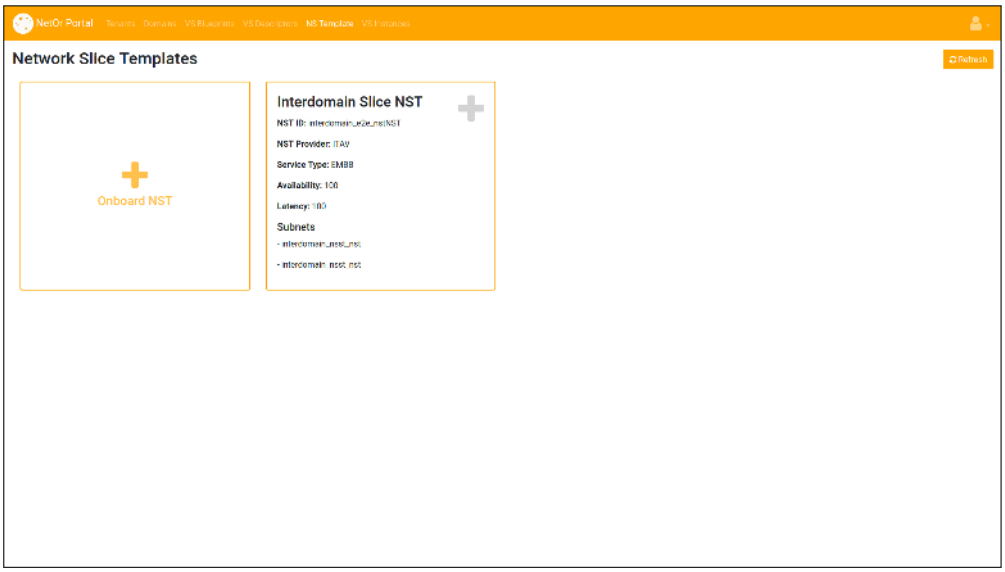


Figure 7: Admin NSTs Page

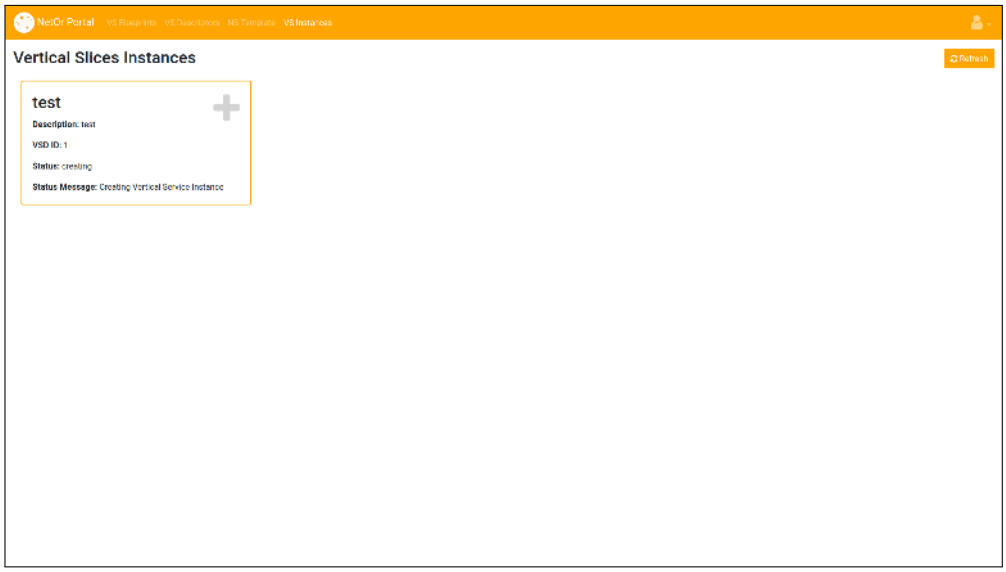


Figure 8: Admin VSIs Page

Tenant Pages

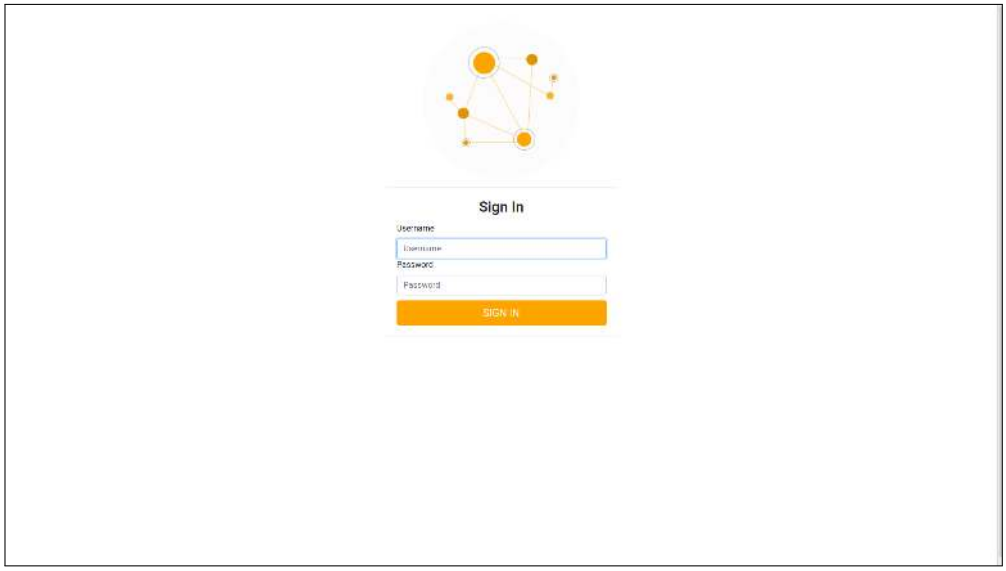


Figure 9: Tenant Login Page

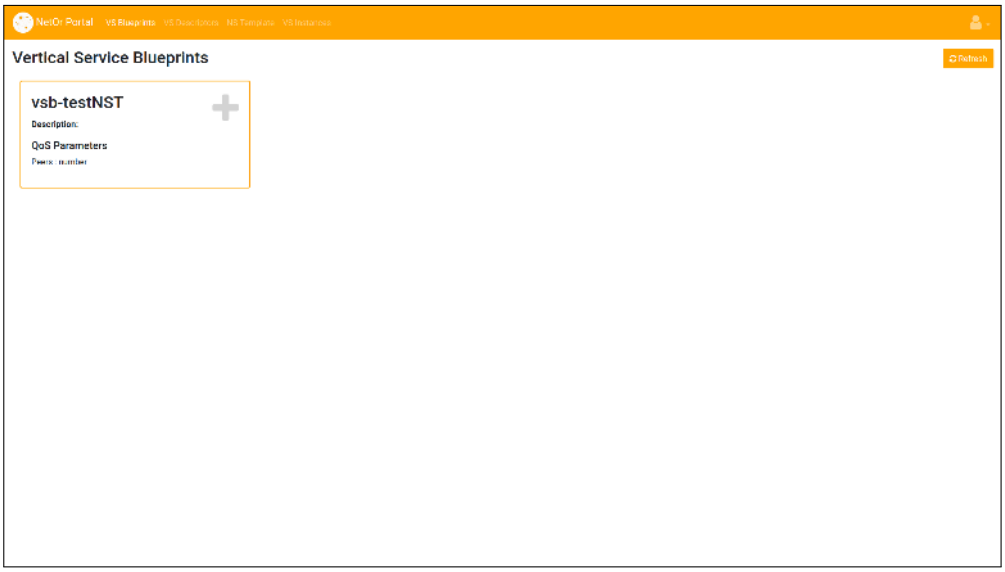


Figure 10: Tenant VSBs Page

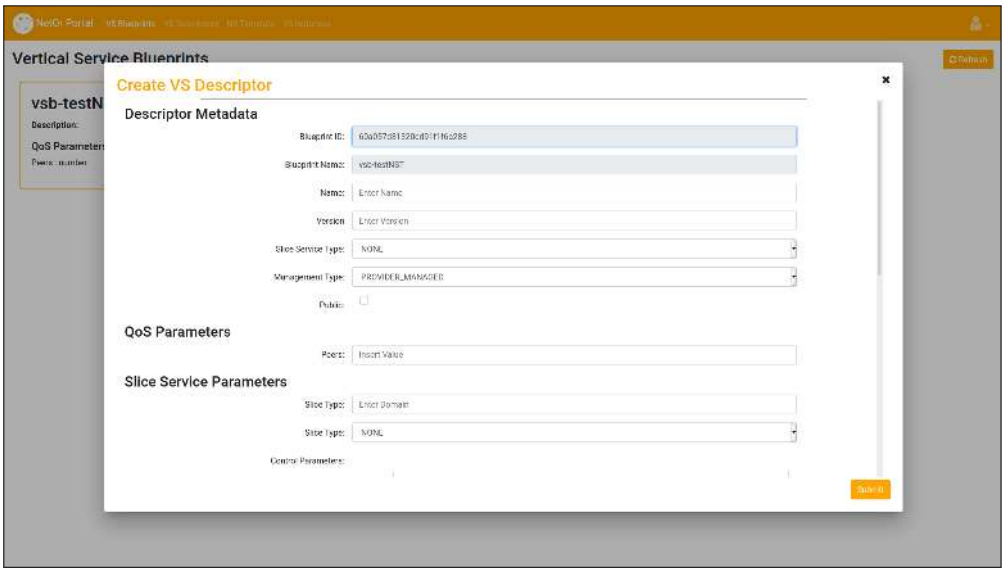


Figure 11: Tenant Create VSD Page

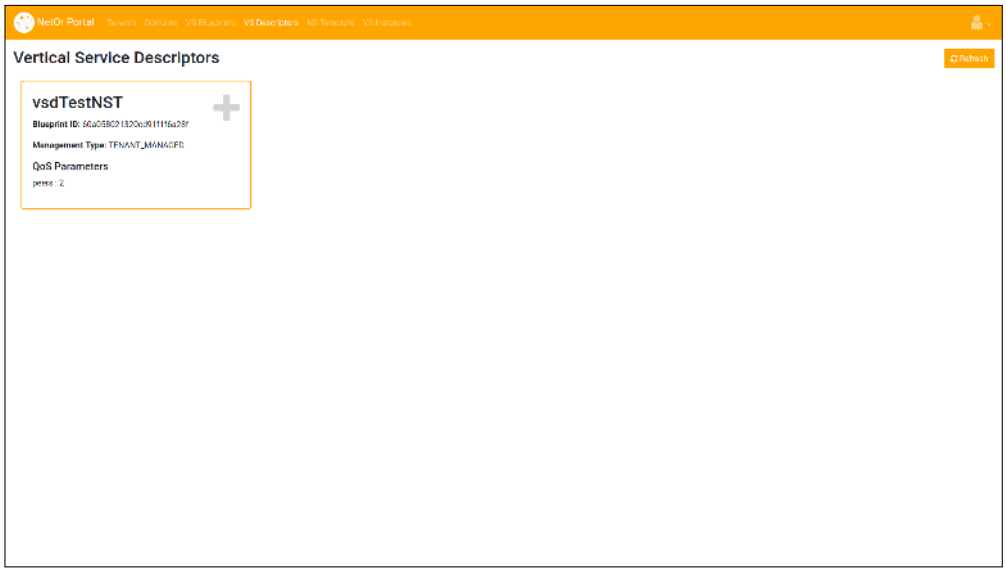


Figure 12: Tenant VSDs Page

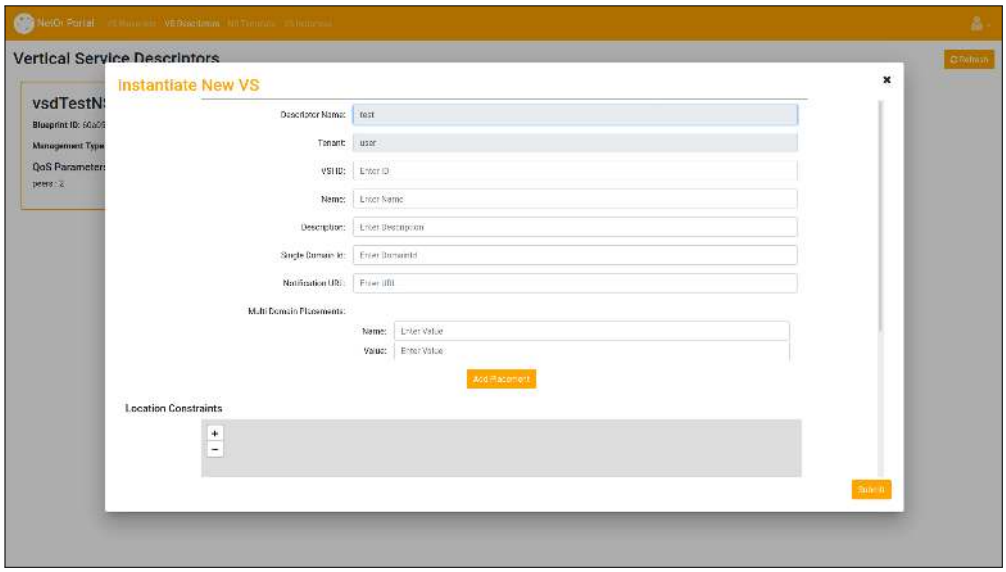


Figure 13: Tenant Instantiate VSI Page

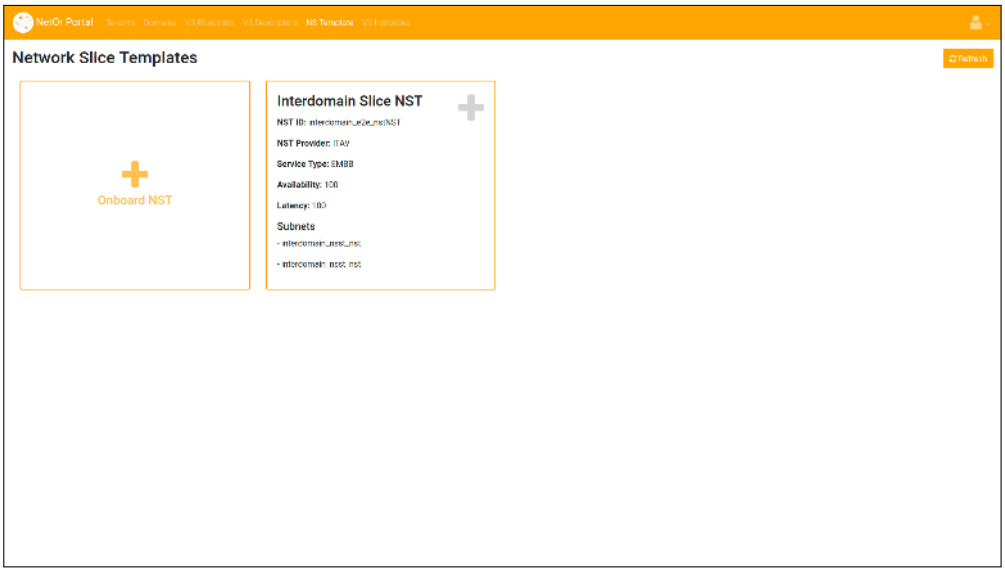


Figure 14: Tenant NSTs Page

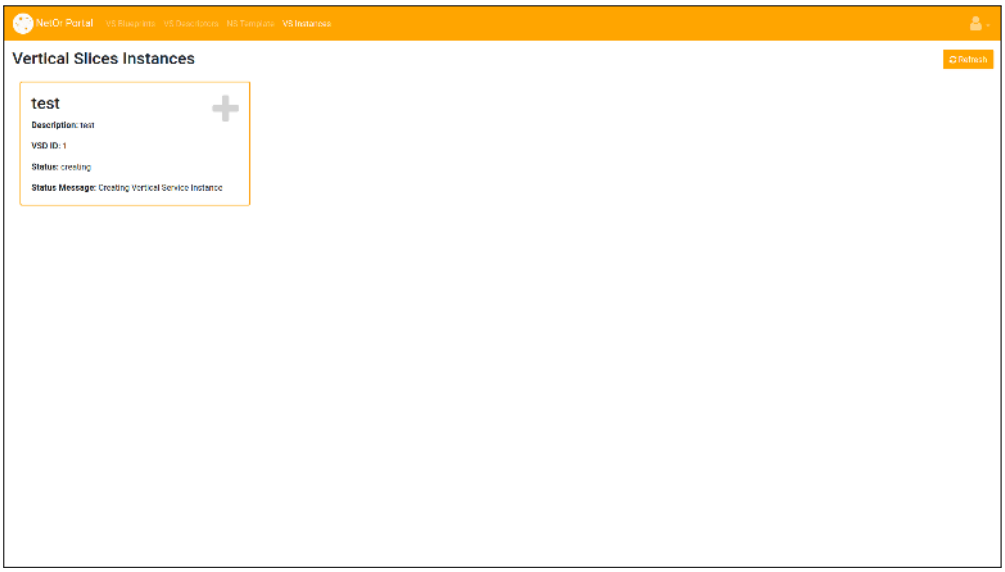


Figure 15: Tenant VSIs Page

Appendix B

Onboarded Vertical Service Blueprint


```
{
    "vs_blueprint": {
        "slice_service_type": "EMBB",
        "embb_service_category":"URBAN_MACRO",
        "parameters": [
        {
            "parameter_id": "peers",
            "parameter_type": "number",
            "applicability_field": "interdomain",
            "parameter_name": "Peers",
            "parameter_description": "#Peers"
        }
        ],
        "version": "version_1",
        "name": "vsbInterdomainTest",
        "inter_site": true
    },
    "translation_rules": [
        {
            "nst_id": "interdomain_e2e_nst",
            "input": [
            {
                "max_value": 5,
                "parameter_id": "peers",
                "min_value": 1
            }
            ],
            "nsd_version": "1.0",
            "blueprint_id": "1"
        }
    ],
    "available_actions": [
        {
            "action_name": "Add Tunnel Peer",
            "action_id": "addpeer",
            "parameters": [
                {
                    "parameter_name": "Peer Network",
                    "parameter_id": "peer_network",
                    "parameter_default_value":"10.100.100.0/24",
                    "parameter_type": "STRING"
                }
            ]
        },
        {
            "action_name": "Fetch Tunnel Peer Info",
            "action_id": "getvnfinfo"
        }
    ],
    "nsts":[
    {
        "nst_version": "1.0",
```

```
        "nsst_ids": [
        "interdomain_nsst_nst_HAL",
        "interdomain_nsst_nst_DEV"
        ],
        "nsst_type": "NONE",
        "nst_service_profile": {
        "service_profile_id": "interdomain_profile",
        "eMBB_perf_req": [
            {
            "user_density": 100,
            "uE_speed": 10
            }
        ],
        "latency": 100,
        "sST": "EMBB",
        "max_number_of_UEs": 1000,
        "availability": 100
        },
        "nsd_version": "1.0",
        "nst_id": "interdomain_e2e_nst",
        "nst_provider": "ITAV",
        "nst_name": "Interdomain Slice NST"
    }
    ]
}
```


Onboarded Vertical Service Descriptor


```
{
    "is_public": true,
    "nested_vsd_ids": {},
    "vs_blueprint_id": "60ccd513ca9b0f7172ba43c9",
    "domain_id": "ITAV",
    "version": "1.0",
    "management_type": "TENANT_MANAGED",
    "name": "vsdInterdomaintest",
    "qos_parameters": {"peers":"2"},
    "tenant_id": "user"
}
```


Onboarded Vertical Service Instance


```
{
    "name":"vsiInterdomainTest",
    "description":"vsiInterdomainTest",
    "vsdId":"60ca601412002d67ff294e67",
    "vsiId":"1",
    "domainPlacements":[
        {
            "domainId":"ITAV",
            "componentName":"1_1-vsiInterdomainTest"
        },
        {
            "domainId":"DETI",
            "componentName":"1_2-vsiInterdomainTest"
        }
    ],
    "additionalConf":[
        {
            "componentName":"1_1-vsiInterdomainTest",
            "conf":"{\"netslice-subnet\": [{\"id\": \"interdomain-tunnel-
            peer\", \"additionalParamsForVnf\": [{\"member-vnf-index\":
            \"1\", \"additionalParams\": {\"use_data_interfaces\":
            \"true\"}}]}]}"
        },{
            "componentName":"1_2-vsiInterdomainTest",
            "conf":"{\"netslice-subnet\": [{\"id\": \"interdomain-tunnel-
            peer\", \"additionalParamsForVnf\": [{\"member-vnf-index\":
```

```
            \"1\", \"additionalParams\": {\"use_data_interfaces\":
            \"true\"}}]}]}"
        }
    ]
}
```