D4.2: Full API and Information Model Specification

Deliverable

Document ID	NECOS-D4.2
Status	Final
Version	1.0
Editor	Panagiotis Papadimitriou (UOM)
Due	31/08/2019
Submitted	08/08/2019

Abstract

This deliverable presents the final version of the NECOS Resource Marketplace related specifications, including the resource discovery and matching methods, the information model, as well as the associated client-to-cloud and cloud-to-cloud APIs. D4.2 further includes more detailed examples of the Marketplace operation, a cost model formulation for resource pricing, and new state-of-the-art descriptions, related to the new contributions. In addition, D4.2 provides evaluations results regarding the performance and operation of the NECOS Marketplace.

TABLE OF CONTENTS

EUB-01-2017

LIST OF FIGURES

CONTRIBUTORS

Contributor	Institution
Panagiotis Papadimitriou	University of Macedonia (UOM)
Lefteris Mamatas	University of Macedonia (UOM)
Ilias Sakellariou	University of Macedonia (UOM)
Sofia Petridou	University of Macedonia (UOM)
Chronis Valsamas	University of Macedonia (UOM)
Sotiris Skaperas	University of Macedonia (UOM)
Antonis Tsioukas	University of Macedonia (UOM)
Sarantis Kalafatidis	University of Macedonia (UOM)
Rafael Pasquini	Federal University of Uberlândia (UFU)
Raquel Fialho Lafetá	Federal University of Uberlândia (UFU)
Fábio Luciano Verdi	Federal University of São Carlos (UFSCAR)
Paulo Ditarso	Federal University of São Carlos (UFSCAR)
André Beltrami	Federal University of São Carlos (UFSCAR)
Javier Baliosian	Universitat Politècnica de Catalunya (UPC)
Fernando Farias	Federal University of Pará (UFPA)
Billy Pinheiro	Federal University of Pará (UFPA)
Antonio Abelem	Federal University of Pará (UFPA)
Christian Rothenberg	University of Campinas (UNICAMP)
David Moura	University of Campinas (UNICAMP)
Asma Swapna	University of Campinas (UNICAMP)
Francesco Tusa	University College London (UCL)
Stuart Clayman	University College London (UCL)
Sand Correa	Federal University of Goias (UFG)
Leandro Freitas	Federal University of Goias (UFG)
Luis M. Contreras	Telefónica Investigación y Desarrollo (TID)

Reviewers

Reviewer	Institution
Alex Galis	University College London (UCL)
Christian Rothenberg	University of Campinas (UNICAMP)
Javier Baliosian	Universitat Politècnica de Catalunya (UPC)

Acronyms

AP	Access Point	PNF	Physical Network Function
API	Application Programming Interface	PNM	Physical Network Manager
B2B	Business-to-Business	RAN	Radio Access Network
C2B	Customer-to-Business	RC	Resource Control
COMS	Common Operation and Management of network Slicing	RDF	Resource Defined Template
ECA	Event-Condition-Action	REST	Representational State Transfer
EPA	Extended Platform Awareness	RT	Resource Topology
ETSI	European Telecommunications Standards Institute	SCM	Slice Charging Management
FIB	Forwarding Information Base	SCPO	Slice Capacity Planning and Optimization
GPS	Global Positioning System	SDO	Service Data Object
IaaS	Infrastructure as a Service	SFM	Slice Fault Management
IM	Information Model	SLA	Service Level Agreement
IMT	International Mobile Telecommunications	SLMCCS	Slice Lifecycle Customer Care Support
ISG	Industry Specification Group	SLO	Service Level Objective
ITU	International Telecommunications Union	SP	Slice Provisioning
KPIs	Key Performance Indicators	SRA	Slice Resource Alternatives
LINP	Logical Isolated Network Partitions	SRMA	Slice Resource Monitoring and Analytics
LSCD	Lightweight Slice Defined Cloud	SSH	Secure Shell
MANO	Management and Orchestration	SSM	Slice Security Management
MdO	Multi-domain Orchestrator	SRR	Slice Resource Repository
ML	Machine Learning	UML	Unified Modeling Language
MPLS	Multiprotocol Label Switching	VDU	Virtual Deployment Unit
NFVO	Network Function Virtualization Orchestrator	VIM	Virtualized Infrastructure Manager

NFV-O	NFV Orchestrator	VL	Virtual Link
NML	Normalized Maximum Likelihood	VNC	Virtual Network Computing
NOVI	Networking Innovations Over Virtualized Infrastructures	VNF	Virtual Network Function
NS	Network Service	VNFCs	VNF Components
NSD	Network Service Descriptor	VNFFG	VNF Forwarding Graph
OAM	Operations and Management	VNP	Virtual Network Provider
OCCI	Open Cloud Computing Interface	VRM	Virtual Resources Manager
OGF	Open Grid Forum	WIM	WAN Infrastructure Manager
OSM	Open Source MANO	XML	Extensible Markup Language
OWL	Web Ontology Language	YAML	YALM Ain't Markup Language
PDT	Partially Defined Template	YANG	Yet Another Next Generation

Executive Summary

The NECOS project aims at the design and implementation of system architecture for cloud slicing across multiple administrative domains. In particular, NECOS promotes cloud slicing in the form of Lightweight Software Defined Cloud (LSDC), which encompasses various novel aspects, such as the ability to define and process slice requests at different granularities and formats, as well as the on-demand instantiation of a Virtual Infrastructure Manager (VIM) per slice, which effectively enables the tenant to exercise fine-grained control and management of his slice.

This deliverable is focused on the NECOS Resource Marketplace and all associated specifications, such as the information model for the specification of the network slices and the physical infrastructure, the APIs that enable the tenant and the physical infrastructure resource provider to interact with the NECOS orchestrator, as well as the various methods for the resource discovery and matching during the processing of slice requests or various run-time operations on slice instances. D4.2 compliments the architecture specification deliverable D3.2 with detailed specifications and examples regarding the operation of the NECOS resource marketplace and its interaction with the rest of the NECOS system architecture components.

The information model provides all required information for the description of network slices and all elements and resources of the physical infrastructure. This facilitates Marketplace operations, such as resource discovery and matching. The information model provides the means to the tenant for slice request specification at different granularities, as well as the ability to address certain slice parts or individual components for various slice run-time operations (*e.g.,* lifecycle management, service deployment). D4.2 provides also a detailed description of a wide range of API methods for (i) slice request, management, and configuration by the tenant, and (ii) slice request, instantiation, and run-time management by the NECOS orchestrator. Both the information model and the cloud APIs have been refined and extended (compared to their initial version documented in D4.1), based on the feedback from the implementation of the NECOS system.

In addition, D4.2 presents a thorough description of the operation of the NECOS Marketplace, with emphasis on methods, information exchange, and system component interactions for the advertising, discovery and matching of resources. These operations are exemplified through detailed examples using YAML. To provide the means for the computation of cost-effective slice instantiations for the tenant, we also introduce a cost model formulation, along with its preliminary assessment based on a range of resource prices. Besides all detailed NECOS Marketplace specifications, D4.2 includes experimental results for the validation of the Marketplace and the assessment of its performance and scalability. This evaluation across multiple experimental infrastructures corroborates the feasibility of the Marketplace and uncovers interesting insights about its performance.

1 Introduction

The NECOS project addresses the challenging problem of network slicing across multiple cloud environments, such as cloud datacenters and edge clouds. In this respect, NECOS aims at building a platform for the provisioning, management, and resource orchestration of network slices, enabling a new cloud computing model, namely ***Slice as a Service***. One of the novel aspects of the project is the on-demand instantiation of a Virtual Infrastructure Manager (VIM) per slice, which effectively enables the tenant[1] to exercise fine-grained control on his slice, eliminating unnecessary provider interventions during the slice lifetime. NECOS supports various slicing operational model, with the current project focus being on VIM-independent slicing (termed as *Mode 0*) and VIM-dependent slicing (termed as *Mode 1*). In particular, *Mode 0* grants the tenant with direct access to a dedicated VIM, whereas *Mode 1* provides a shared VIM among multiple tenants.

This deliverable provides a detailed description of the **NECOS Marketplace** and all associated specifications, including updates of the D4.1 **Information Model** and **Application Programming Interface** (API) specifications for slice request, provisioning, and run-time management. In this respect, D4.2 elaborates on the Marketplace operations, including all methods, information exchange, and system component interactions for resource advertising, discovery and matching. Resource discovery and matching involve various NECOS components, such as the *Slice Builder*, the *Slice Broker*, and the *Slice Agents*. These operations are explained through detailed examples using YAML. In the context of the Marketplace, we further present a cost model formulation, along with its preliminary assessment based on a range of resource prices.

Slice creation raises the need for means to describe slice requests as well as physical resources. This inherent need is satisfied through an information model that provides resource descriptions at different levels of granularity, meeting the requirements of slice specifications and infrastructure description. This provisional information model has been developed after a careful inspection of related information models, such as COMS (Common Operations and Management on network Slices) and ETSI NFV MANO. The deliverable provides a refined and extended description of the NECOS information model, based on the NECOS system implementation.

The deliverable further reports on a set of cloud APIs to enable slice request, creation, configuration, and run-time management. The respective cloud APIs have been subdivided into two classes: (i) client-to-cloud APIs, which include API methods invoked by the tenant for slice request as well as slice management and control upon the slice creation, and (ii) cloud-to-cloud APIs, which are associated with interactions between NECOS system components residing in different domains (*e.g.,* in the case of cloud federation), such as the *Slice Resource Orchestrator*, the *Slice Builder*, *Slice Broker*, the *Slice Agents*, and the *Slice Controllers*. More specifically, the set of *Cloud-to-Cloud APIs* comprises the following APIs: (i) *Slice Request Interface*, (ii) *Slice Instantiation Interface*, (iii) *Slice Marketplace Interface*, and (ii) *Slice Runtime Interface*. Similar to the information model, all cloud API specifications have been refined and extended, exploiting the feedback from the NECOS system implementation.

Besides all detailed specifications, the deliverable also includes an updated state-of-the-art analysis (SOTA) that encompasses the following areas: (i) information models, (ii) APIs, (iii) Marketplaces, and (iv) cost models. As such, we clarify the contributions and novel aspects of the NECOS Marketplace with respect to other projects and initiatives. Finally, experimental results for the validation and the assessment of the Marketplace are presented. The Marketplace evaluation study across multiple experimental infrastructures shows that resource discovery and matching incur low delays and also yield nice scalability properties.

[1] The terms *tenant* and *client* are used interchangeably, representing the cloud service consumer (*i.e.,* Slice-as-a-Service consumer, in the context of NECOS).

1.1 Deliverable Structure

The deliverable structure provides a clear separation between the three main outputs of NECOS WP4 (*i.e.,* Marketplace operations, information model, cloud APIs) and the state-of-the-art (SOTA), helping the reader to grasp the project contributions and further understand how the project goes beyond the SOTA in the area of cloud network slicing.

In further detail, the deliverable is structured as follows. Section 2 provides a SOTA of relevant cloud APIs, information models, Marketplace specifications, and cost models. After the analysis, the deliverable identifies the gaps and extracts the useful features for network slicing. Section 3 provides a detailed documentation of the information model for the slice specification and the infrastructure description. Initially, there is a high-level representation of the whole model, followed by more detailed descriptions of (i) the slice, as specified, requested and viewed by the *Tenant*, and (ii) the physical infrastructure, which includes very detailed specifications of the main infrastructure components for resource availability, allocation and monitoring by the infrastructure provider. Section 4 elaborates on the Marketplace operation, as well as on the interactions and information exchange between the involved NECOS system components for resource discovery and matching. Section 4 presents a cost model formulation with its preliminary assessment, as well as an initial experimental study of the Marketplace. Section 5 documents the two classes of cloud APIs (namely *client-to-cloud* and *cloud-to-cloud*), providing a description of all supported API methods. Each API class is presented in a separate subsection. Finally, Section 6 provides a summary of the project contributions with respect to the Marketplace operation. The deliverable also includes an Appendix with more detailed SOTA of cloud APIs and information models.

1.2 Contribution of this Deliverable to the project and relation with other Deliverables

This deliverable documents the final outputs of WP4 and, more specifically: (i) detailed descriptions of the operations, methods and system component interactions for resource advertisement, discovery, and matching, all being part of the NECOS Marketplace, (ii) an information model for slice specification and infrastructure description at different granularities, and (iii) a wide range of API methods for slice request, creation, configuration and run-time management. These contributions complement the NECOS slicing architecture, which is presented in D3.2. In particular, D4.2 provides the necessary means for slice specification and provisioning, based on the architecture that appears in D3.2. This deliverable will further provide inputs to D5.2 and D6.2, and more specifically, to the feasibility and performance tests that will be conducted based on the integrated NECOS system implementation.

In relation to the previous version (*i.e.,* D4.1), this deliverable includes the following extensions and improvements (amongst other refinements, not listed below):

- A description of the resource matching mechanism used to match requested to advertised resources, using rules (Section 4.2).
- The mathematical formulation of a cost model, under which Infrastructure Providers annotate their resource offers (Section 4.3).
- Experimental results and assessment of the main Marketplace operations, *i.e.,* resource discovery and matching (Section 4.4).
- Description of Service Level Objectives (Section 3.2.1), policies (Section 3.2.2), and monitoring parameters (Section 3.2) in the information model.
- Slice monitoring API methods that enable either the tenant or the Slice Resource Orchestrator to interface with a monitoring subsystem instantiated on a slice (or slice part).
- Improved examples of Marketplace message exchanges that relate to the advanced touristic service presented in Section 3.4.
- New SOTA areas on cost models (Section 2.3) and Marketplace specifications (Section 2.4).

2 State-of-the-Art Analysis

During its second year, NECOS evolved its architecture from a use-case driven end-to-end slicing architecture to an experimentation prototype, exercising extensively its novel features. The architectural artefacts defined in D4.1 (*e.g.,* information model, APIs, marketplace-based multi-domain resource discovery, etc.) have been validated in practice and enriched. This crucial feedback is a main input for this deliverable. In this respect, we have extended the state-of-the-art analysis (SOTA) of the relevant platforms and architectures to NECOS with recent approaches, including those enabling real experimentation. Our direction is aligned to the advancement of the 5G networking technologies to their trial phase, targeting a commercial deployment in two years [ALLIANCE20155g].

A number of recent proposals support service-driven E2E slicing and experimentation. 5G platforms with inherent 5G E2E slicing capabilities include: (i) the 5G-VINNI targeting particular 5G KPIs and multiple verticals; (ii) the 5G-PAGODA [PAGODA2017] investigating NFV-based E2E slicing over two test-beds in Europe and Japan; (iii) 5G-EVE which is an experimentation-oriented platform for multiple verticals, supporting a cross-facility 5G catalogue and multi-domain orchestration. Other proposals focus on RAN slicing aspects (*e.g.,* [RAN]); and (iv) SliceNet [SLICENET] investigating the management and control planes of network slicing across multiple administrative domains. The 5G-MEDIA [5G-MEDIA] project and platform targets the Media Verticals through multiple use-cases, focuses on SDN/NFV aspects and supports a DevOps environment for media applications, hiding the complexity of service development and deployment over 5G infrastructures.

A number of related works build on top of novel federated test-bed facilities, such as FED4FIRE [wauters2014federation] and GENI [GENI]. Indicative examples are: (i) FUTEBOL [FUTEBOL] integrating wireless and optical domains over European and Brazilian test-beds; (ii) SoftFIRE, an SDN/NFV test-bed supporting high-level service definition based on TOSCA [TOSCA] and resource discovery; (iii) 5GinFIRE [5GINFIRE] a 5G test-bed targeting multiple vertical industries; and (iv) 5G-CDN [5G-CDN] testing main NECOS ideas towards implementing slicing for next-generation Content Distribution Networking (CDN) services. Such test-beds support slicing features, including RAN slicing (e.g., the LTE slicing feature of the NITOS test-bed [NITOS-RAN]).

Along these lines, we utilize in NECOS the novel FED4FIRE facilities [WAUTERS2014], offering: (i) a representation of heterogeneous hardware resources around the globe including, but not limited, physical servers and networking equipment located in the multi-domain infrastructure providers formulating the federated FED4FIRE test-beds; (ii) a basis for end-to-end network slicing in the sense of putting together physical resources from different test-beds, called slivers, to implement a particular service for experimentation; (iii) automations in the resource discovery and allocation through the FED4FIRE testbed control tools (*e.g.,* jFed CLI).

In the following, we discuss four SOTA areas related to the main D4.2 artefacts, *i.e.,* cloud APIs, information models, cost models, and marketplace specifications.

2.1 Cloud APIs

Regarding the Cloud APIs, a number of relevant efforts in research projects and other initiatives have been investigated towards extracting useful information that can be fed into the respective specifications of NECOS. Indicatively, we take into account the *5G-PPP* proposal regarding the 5G architecture and the functionalities supported, *e.g.,* network slicing, softwarization, management and orchestration; the *5GEx* architecture framework; the *ITU-T* conceptual architecture for network virtualization that includes specific functional components for network slice lifecycle management and orchestration, and network slice instance management; and the *Open Grid Forum* (OGF) with the relevant Open Cloud Computing Interface (OCCI).

The 5G-PPP assumes "the ability to build a service out of existing services" defining a recursive approach in network slicing concept. More specifically, a tenant can operate its virtual infrastructure as it operates the physical one, allocating and reselling part of the resources to other tenants in a recursive manner. As such, each tenant can possess and deploy its own MANO system [MANO]. To support such functionality, a set of homogeneous APIs has been defined to provide an abstraction layer for the slice management and the underlying virtual resources' control. In respect to the network slicing, 5G-PPP proposes a set of APIs for the interaction between Network Services (NS) and the corresponding VNFs encompassing attributes such as network-slice ID, nodes, links, storage and compute resources, network services, and functions and virtual network functions, among others. In addition, 5G-PPP proposes a general categorization of APIs as a means to express the different degree of network slices' control, *i.e.,* network service allocation / modification / de-allocation API, virtual infrastructure allocation / modification / de-allocation API, virtual infrastructure control API with limited control, and, virtual infrastructure control API with full control. Finally, the 5G-PPP also considers APIs for multi-domain orchestration.

For ITU-T, network slicing is perceived as Logical Isolated Network Partitions (LINP). According to the Recommendation ITU-T Y.3011 [ITU-T Y.3011], a LINP is composed of multiple virtual resources, whose capability may be not bound to the capability of the physical or logical resource, which is isolated and equipped with a programmable control and data plane. Thus, network virtualization is seen as a method that allows multiple LINPs to coexist in a single physical network. In April 2018, ITU-T released Draft Recommendation ITU-T Y.3112 (Y.IMT2020-MultiSL) - Framework for the support of Multiple Network Slicing. This Recommendation describes the concept of network slicing and use-cases of multiple network slicing, enabling a single device to simultaneously connect to different network slices. The use case describes the slice service type for indicating a specific network slice and the slice user group for precisely representing the network slice in terms of performance requirements and business models. Finally, it also specifies the high-level requirements and high-level architecture for multiple network slicing in IMT-2020 network (International Mobile Telecommunication). However, APIs for network slicing in clouds is still a pending issue for ITU-T IMT-2020 future evolvements.

The OGF working groups have been studying several proposals, and the most relevant to NECOS is the OCCI, which focuses on the cloud computing IaaS based model. OCCI is a protocol and API that aims to enable the development of interoperable tools for common tasks, including deployment, autonomous scaling, and monitoring. OCCI is able to abstract and generalize methods or call specific functions of a particular VIM (or any other management software). However, this API does not support network slicing in terms of proceeding with slice instantiation or monitoring.

The NECOS project takes advantage of the 5GEx proposal [5GEX], which defines three main network interfaces: (i) between the customer and the multi-domain orchestrator for service exposure (B2C), (ii) for multi-domain orchestrators' interaction (B2B), and (iii) for the interaction between the multi-domain orchestrator and the domain orchestrators within the same administrative domain. In accordance with this scheme, NECOS proceeds with defining *Client-to-Cloud APIs* and a set of interfaces composing the *Cloud-to-Cloud APIs*, which will be enhanced with slice-related methods for slice provisioning and run-time management.

A more extensive review of relevant cloud APIs is available at the Appendix of this deliverable.

2.2 Information models

The NOVI project [NOVI2015] defined an architecture for supporting federation of infrastructures. The project has specified a common Information Model (IM) as the essential element to achieve the federation goals. Their model was intended to support virtualized resources and context-aware resource selection; to be vendor-independent; to support monitoring and measurement concepts, and to support management policies. NOVI uses Web Ontology Language (OWL) for modelling virtualization explicitly for both, computing and networking devices using Web Ontology Language (OWL). It is vendor-agnostic, modular, and composed of three main ontologies: (i) resource ontology, (ii) monitoring ontology, and (iii) policy ontology.

The European Telecommunications Standards Institute (ETSI) has defined a framework for Network Functions Virtualization (NFV) and Management and Orchestration Architectures (MANO) [MANO]. Information in a network service (NS) is structured into information elements, which might contain a single value or additional information elements that form a tree structure. Information elements are classified as one of the following types: leaf element (single information element), reference element (information element that contains a reference to another information element) and sub-element (information element that specifies another level in the tree). The information elements can be used in two different contexts: as descriptors or as run-time instance records. A descriptor is defined as a configuration template that defines the main properties of managed objects in a network. The network service descriptor (NSD) is the top-level construct used for designing the service chains, referencing all other descriptors that describe components that are part of that network service. Additional descriptors are used by MANO for the following information elements: virtual network function (VNF), the physical network function (PNF), virtual link (VL) and VNF forwarding graph (VNFFG).

The 4WARD project [4WARD] designed an architecture for the provisioning and management of service-tailored virtual networks across multiple administrative domains. 4WARD relies on a centralized coordinator, namely Virtual Network Provider (VNP), for virtual network deployment. In this context, the project has specified an information model for the description of the infrastructure and the virtual network resources. The information model is used by all actors to specify and exchange virtual/physical resource information during resource advertisement, assignment, monitoring, and allocation of virtual networks. The model uses the abstraction "Network Element" to describe nodes, interfaces, links, and paths. The information model ensures the binding of different elements, such as the binding of links with paths, interfaces with nodes, and so forth. However, in the context of network slicing, the specific model exhibits considerable limitations. First, there are not sufficient elements and attributes in the model to express service elements (*e.g.*, vNFs) as well as vNF graphs. Hence, this model cannot support the slice specification requirements in all NECOS modes (*e.g., Mode 3,* which is associated with service-oriented slice requests). Furthermore, the 4WARD information model does not include descriptors for dedicated infrastructure elements, such as switches, routers, Wi-Fi access points, and base stations. Another limitation of the model is the lack of support for extended platform awareness (EPA), *i.e.,* the exposure of certain capabilities of the infrastructure to the tenant for slice deployment.

The Common Operation and Management of network Slicing (COMS) [ietfcoms2018a] aims at providing a comprehensive approach for the overall operation and management of network slicing, for both network slice operators and network slice tenants. Working on the top network orchestrator inside Transport Network region, which directly communicates with the network slice provider, COMS enables technology-independent network slice management [ietfcoms2018b]. In this context, COMS provides a technology-independent information model for transport network slicing [ietfcoms2018c]. The COMS information model uses the data model for network topologies as a base [ietfdata2018] and enhances it with new slice-specific attributes under the "*netslice*" namespace. COMS uses the YANG data modeling language [RFC7950] to make a technology-independent representation of the transport network slice data-model. This information model includes information elements, such as compute resources, connectivity resources, storage resources, service instances, network slices. COMS further supports operations on slices, which map to NECOS, but they need to be extended to accommodate its different slicing modes enabled by NECOS.

Further details on these information models can be found in the Appendix.

2.3 Cost models

In the cloud and grid computing area, cost analysis has been studied for over a decade now (*e.g.,* [ALTMANN2008, KONDO2009]). The authors in [ALTMANN2008] present the rationale for a grid market and, the introduction of a marketplace for trading commoditized computing resources. The proposed marketplace offers computing resources from different providers through virtualization. The proposal includes the definition of a spot and future market as parameters that should be considered as a market mechanism for computing resources.

Authors in [KONDO2009] develop a cost-benefit analysis of deploying volunteer computing desktop grids on the Amazon AWS public cloud computing infrastructure. A cost model for the volunteer computing scenario is described, based on the experiences with two mid-to-big size projects, namely:

XtremLab and SETI@home. The authors suggest that the costs of maintaining such projects on a cloud computing environment are prohibitive, despite the elastic characteristics of the cloud.

In addition, cost models are used by researchers of different fields in cloud computing, from proposals for data management in the cloud [NGUYEN2012] to the composition of virtual networks in multiple domains [DIETRICH2013].

Authors in [NGUYEN2012] propose an approach for decreasing the cost of data management in the cloud, by using a classical database performance optimization technique, namely view materialization. Their contributions include novel cost models that complement the existing materialized view cost models with a monetary cost component that is primordial in the cloud. Subsequently, these cost models are used into an optimization process that achieves a trade-off between computing power enhancement and view materialization, taking into consideration budget constraints. The experimental validation hints that view materialization is always desirable.

Authors in [DIETRICH2013] evaluate the challenging problem of multi-domain virtual network embedding with limited information disclosure, considering that virtual network providers have a limited view on substrate network resources. As stated in the paper, the main contributions are twofold: a traffic matrix based virtual network embedding framework that enables request partitioning; and a feasibility study on the embedding process with limited information disclosure, compared to a best-case scenario where all information is available to virtual network Providers.

The pricing and negotiation schemes are important aspects of a marketplace. The predominant pay-as-you-go models, as those employed by cloud providers for years, have to move towards a more complex computational economics in order to favour federation. New models provide a way to reach agreement while representing the diverse perspectives of consumers and providers. In fact, several economic protocols have been proposed for computational resource allocation. In particular, auctions have been proven to be a popular protocol for computational economies. Auctions schemes facilitate negotiation and are able to efficiently establish a market price. Examples, such as [BARANWAL15, KUMAR2017, KUMAR2018, TAFSIRI2018], all provide support for auction-based allocation. In common, these works consider VMs and their attributes as the resources being traded between providers and customers (users/clients/tenants), and an auctioneer component, such as a middleware broker. In general, the proposals seek for trustworthiness, fairness, economic and allocation efficiency; the problem is formulated as a Linear Programming method; an evaluation is carried out by simulation (other studies, *e.g.*, [TAFSIRI2018] also propose a heuristic-based algorithm as an alternative for running on-premises). Although these solutions are based on well-established trading mechanisms like auctions, they do not take into account the whole spectrum of resource types and attributes in an environment, such as cloud network slicing. A deeper investigation on the feasibility of using auction-based pricing models on NECOS Marketplace is highly desired, considering the multi-domain, resources, and attributes of NECOS ecosystem. New proposals have to be motivated by the necessity in providing more flexible and efficient marketplaces.

Although many auction solutions have been proposed, this kind of market strategy seems more suitable for services than for physical resources, since services are more susceptible to the price fluctuations.

Authors in [OLOUGHLIN2018] try to address the problem of performance variation across instances of the same type in the trading of virtual machines, storage, and other cloud computing resources. To do so, they propose a Cloud Service Broker that re-sells instances based on current hardware performance indicators. The strategy is to re-price instances according to their deliverable performance. This secondary market is analysed through simulations, from which the authors conclude that hardware heterogeneity is beneficial to the Cloud Service Broker, since it increases the offers, among other results.

[CHARD2019] describes a cloud federation approach that enables resource requests to be allocated to any set of infrastructure providers. The authors advocate the idea of exposing the cost models via an open market, in which tenants and providers are matched considering a trade-off between capabilities and cost. A co-operative infrastructure is proposed, where the management and operations are hosted on the cloud providers themselves, turning these tasks both ownership and operator agnostic. Despite the proposal also been said protocol- and workload- agnostic, authors analyse the application's security and privacy in protecting auction protocols for resource allocation.

Authors in [CHUNLIN2019] investigate the resource management problem based on load balance in edge and cloud environments. More specifically, they address the problems of resource granularity of the cloud service provider and the data loss that resource shrinkage might bring in edge scenarios. The paper proposes a resource expansion and shrinkage model based on the service cost to decrease the cost of edge cloud clusters, at the same time cluster load condition is satisfied. Furthermore, a data migration model is also proposed in order to deal with the problems of unbalanced cluster load after an expansion and data reliability after a shrinkage. Numerical results show evidence that the proposed algorithm can decrease the cost of edge cloud cluster and guarantee both data integrity and load balancing.

Authors in [WU2019] present an extensive study on value-based pricing modeling for the cloud market. The paper describes a framework of value-based pricing strategy that takes into account the theory of value co-creation for both customers and providers in order to form business partnerships. Moreover, the authors show how to create different pricing models and, for each one, demonstrate how to identify the optimal price point, which maximizes providers' profit by using a genetic algorithm. The work also presents a comprehensive state-of-the-art investigation on cloud pricing for different business to business cloud market segments.

2.4 Marketplace

[FENDE] introduces a marketplace ecosystem where users can discover and execute virtualized network functions (VNFs) and compose service function chains (SFCs). The proposed architecture considers three classes of users, namely the developers, reviewers, and customers, as depicted in Figure 1. Each class interacts with FENDE through dedicated access and management panels that provide all operations needed for marketplace operations. [FENDE] provides infrastructure support to execute and monitor virtualized functions, and thus, it is a solution that combines marketplace, management, and infrastructure capabilities.

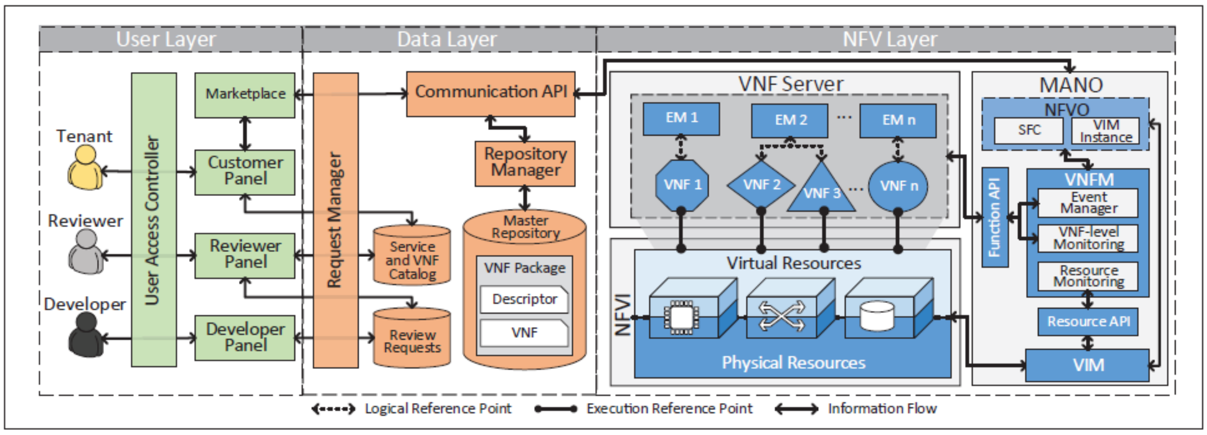


Figure 1. FENDE architecture (Source: [FENDE]).

The architecture consists of three layers, namely the *User*, *Data,* and *NFV Layer*. The main mechanism for VNF discovery presents similarities with that of Google Play and Apple Store, *i.e.,* a trusted (reviewed) repository of VNF's that users instantiate and execute on the provided infrastructure layer. More specifically, developers at the user layer submit requests for VNFs to be offered specifying characteristics, such as source code and virtualization requirements. Once these requests are being approved by the reviewers, the VNFs become available to the tenants through the FENDE marketplace. The tenants have access to their own library containing acquired VNFs, they can create SFCs over them, while they also have the ability to perform life cycle management operations. The data layer maintains three databases to handle all relative to VNFs and SFCs information. Indicatively, a catalogue serves as a menu of the available VNFs and SFCs in the FENDE marketplace, while a master repository keeps track of the instantiated VNFs. Finally, the NFV layer provides the required functionality which regards the services' and functions' lifecycle management (*e.g.,* instantiating, removing or updating VNFs, creating SFCs), resource sharing among virtual elements, control, and orchestration. FENDE supports composing NFVs over multiple network domains and declares intelligence in service provisioning and composition process. However, as opposed to the NECOS vision, it does not implement elasticity mechanisms. NECOS recognises and implements two types of elasticity capabilities: vertical and horizontal. Vertical elasticity in the NECOS marketplace is associated with the resources inside slice parts (*e.g.,* addition or removal of computing, memory or storage) or network parts, while horizontal elasticity is applied when new slice parts are needed. In addition, the NECOS marketplace supports the process of discovering physical resources that meet specific compute and connectivity constraints and proceed with slice instantiation to deploy tenant services.

In T-NOVA [TNOVA2014, TNOVA_TNSM], the notion of a marketplace is used in a very similar to FENDE vein, *i.e.,* a repository where a variety of developers can publish their VNFs offered to customers. As illustrated in Figure 2, apart from the core NFVI operations, *i.e.,* virtualization, orchestration and resource management, T-NOVA implements a *Function Store* following the path of the successful OS-specific app stores [TNOVA2014]. The store consists of network functions provided by several third-party developers, and through a customer front-end is accessible by the clients who browse and place their requests for virtual appliances, facilitating the composition of network services out of the selected VNFs. To enable the submission of service requests from clients, T-NOVA defines a description model for VNFs and network services. T-NOVA also offers a brokerage module through which the clients describe their NFV requirements, receive back offers and proceed with decisions that match their needs and take into account the billing models and the associated Service Level Agreements (SLAs).

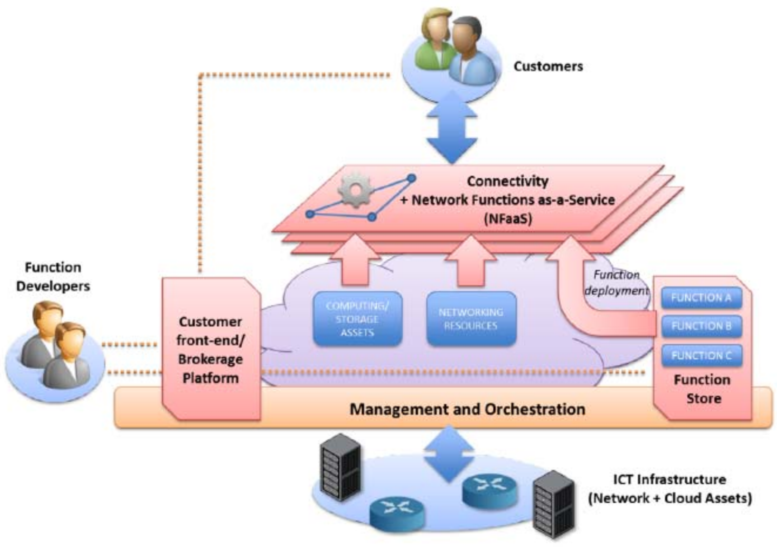


Figure 2. An abstract view of the T-NOVA architecture (Source: [TNOVA2014]).

Pricing mechanisms are an asset of the T-NOVA marketplace supporting different policies spanning from short- and long-term lease to scheduled lease or spot markets. However, the brokerage module that carries out the selection of VNFs can only access Points-of-Presence (PoP) from a single NFV provider. Since T-NOVA does not account for multi-provider NFV deployments, the applicability of its marketplace has a limited scope compared to the proposed Marketplace of NECOS.

5G!Pagoda [PAGODA2017] proposed a 5G architecture that enables the orchestration, instantiation and management of end-to-end network slices both over a single administrative domain as well as across multiple domains. Similarly to the NECOS project, slice creation begins with specific application/service requirements described in the tenants' requests, while slice provisioning utilizes major technological enablers such as SDN, NFV, and cloud computing. The architecture considers a hierarchical fashion regarding resource orchestration and it is service-oriented in the sense that it aims at mapping different slices to virtual resources. Although it leverages the ETSI NFV architecture to cover the orchestration of physical resources across multiple domains, it does not address dynamic discovery over physical resources. Moreover, 5G!Pagoda does not elaborate on the marketplace aspect and issues tailored to that, *e.g.*, cost/billing models and SLAs.

The 5G NORMA project [5GNORMA2017] proposes a business model of network slicing with an optimized 5G infrastructure market [5GNORMAd5.1]. The project introduces an admission control schema accepting new slices to the market based on the resource availability, while bringing network infrastructure providers and the network slices' tenants under a marketplace ecosystem [5GNORMAd7.2].

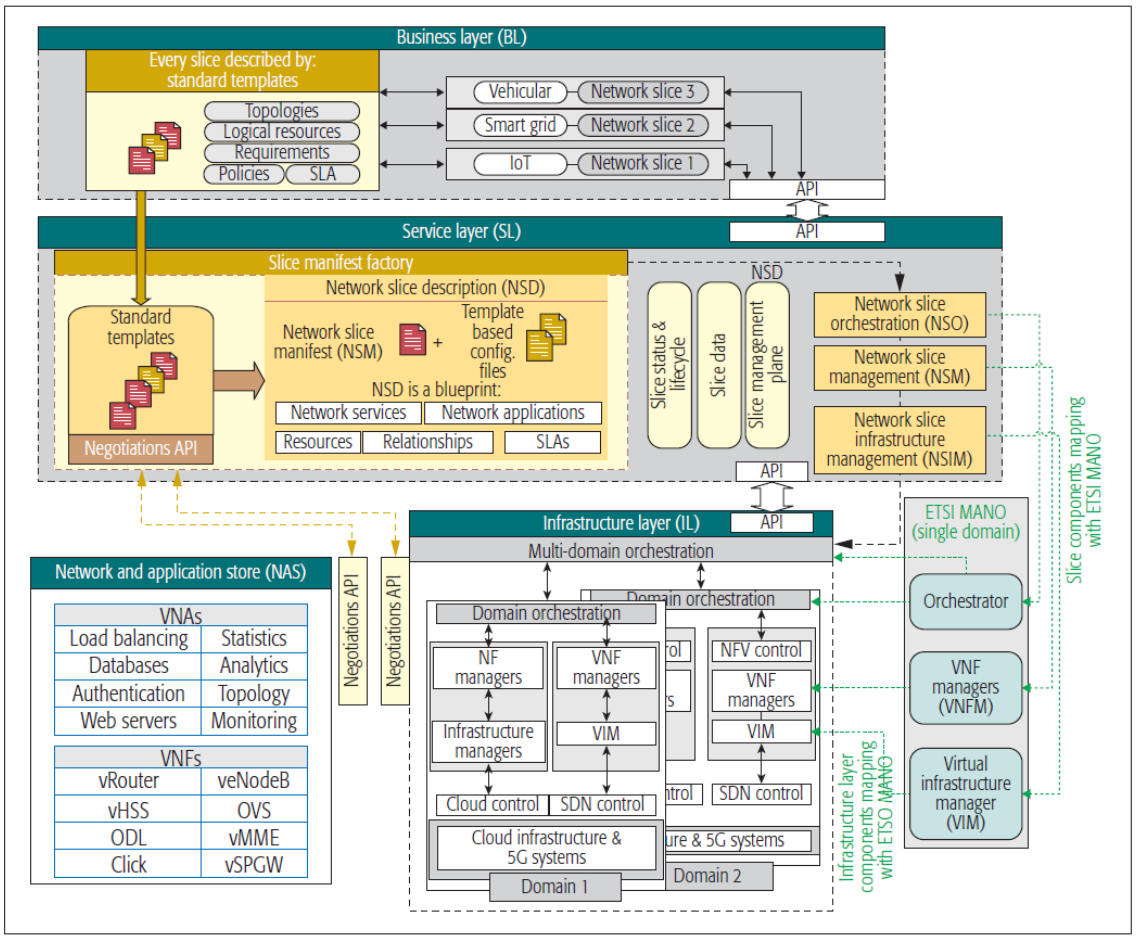


Figure 3. Network Slicing Architecture towards 5G communications (Source: [5GNORMA2017]).

The network slicing architecture proposed in the 5G NORMA project consists of three layers, namely the Business Layer (BL), the Network Slice Service Layer (SL) and an Infrastructure Layer (IL), together with the operation of Network and Application Store (NAS) (Figure 3), which brings similarities with the Service and VNF Catalog of FENDE [FENDE] and the Function Store of TNOVA [TNOVA2014, TNOVA_TNSM]. In 5G NORMA, the authors claim support of complex use-cases on per Network Slice basis, which can be offered by coupling of a rich set of (VNFs) and Virtual Network Applications (VNAs) found on NAS. They envision a multiple stores NAS exposing them to the SL using a common API. The marketplace platform of 5G NORMA follows a vertical marketing model and constitutes of two players: (i) the infrastructure providers (IPs) offering on-demand the slice resources; and (ii) the tenants with rights to acquire requested network slices [5GNORMA2017]. The concept of 5G NORMA's business model focuses on resource allocation mechanisms by controlling market players', *i.e.,* IPs and Tenants, admissibility into the market with a view to enhance performance and revenues of both of the parties. 5G NORMA introduces the notion of a brokerage model residing at the infrastructure providers, multi-tenancy, multi-service support and a controlled access to the market. In contrast to the 5G NORMA, the NECOS marketplace proposal enables dynamic slice deployment and elasticity over multi-domain resources offered by demand from different infrastructure providers, based on the service and cost requirements of the slice user or tenant.

ONAP [ONAP] introduces a reference implementation of a marketplace in the form of VNF, SDKs and APIs in providing policy-driven orchestration and automation of physical and virtual network functions. The model incorporates two actors inside the marketplace platform: i) the vendors, ready to provide VNF and network services; and ii) the operators, capable of performing queries on VNFs already available in the marketplace. Once uploaded by the vendors (or suppliers), each VNF is validated within the marketplace platform automatically and therefore is enlisted as available in the market for purchase. The marketplace provides an on-boarding report for each VNF, for the operators to test and validate them before the purchase [ONAPwiki]. The orientation of ONAP's marketplace architecture is rooted on a catalog model, as the orchestration platform provides an Active and Available Inventory (AAI) model based on resource cartography. AAI stores information regarding the resources and services of a single provider and tenant present in the marketplace [ONAP2018]. Unlike ONAP, the NECOS marketplace considers resource requests and their matching resources from a diverse set of tenants and providers while being an essential part of the slice deployment and operation process.

The 5GTANGO [5GTANGO] project proposes an integrated vendor-independent platform, where a packaged NFV forwarding graph of composed services is automatically tested and validated. In 5GTANGO the marketplace appears as the concept of a store with a customizable orchestrator, network slice manager and slice-to-network-service-mapper, compatible with common existing Virtual Infrastructure Managers (VIM) and slice controllers [5GTANGO-1]. Taking into account vertical application requirements and defined SLAs, 5GTANGO follows complex resource allocation schemes, fitted into dedicated network slice blueprints that are suitable for vertical industries. The primary players are indicated as the Service/Infrastructure Provider, the end-user that could be a developer of VNFs or network services (NSs), or/and the service consumer [5GTANGO-2]. The proposed framework employs artificial neural networks (ANNs) and acts as a mediator between the service providers and the end-users [5GTANGO-3]. In our case, the NECOS marketplace approach fundamentally implements a resource discovery schema between Infrastructure Providers and Slice Providers leveraging a brokerage model inside the platform.

Finally, the SONATA project [SONATA2017] introduces a service platform for both providers (*i.e.,* network operators) and developers through an integrated DevOps model to locally prototype and test complete service function chains [SONATA2017]. The tight integration between the two main building blocks of SONATA architecture (*i.e.,* the service platform and the software development toolkit) offers flexibility and convenience in adding management and orchestration functionalities on-the-fly. The SONATA service platform introduces a brokerage system that exchanges information between the loosely coupled components of the platforms (*i.e.,* aligned to the Micro-services paradigm) [SONATAd4.3]. The status information on the running network services and functions is held in a catalog system accessed by the brokers and stakeholders of the service platform. However, the idea of a generic marketplace and the direct communication between the network operators and the developers are beyond the project's scope.

2.5 Summary

In summary, the context of Slice-as-a-Service, as promoted by NECOS, requires the specification of an Information Model that will describe both the infrastructure resources/elements along with their properties, as well as the slice components and the service elements deployed on top of them. The

NECOS information model should exploit significant features of state-of-the art models, while overcoming limitations mainly related to network slicing and multi-domain physical infrastructures.

The NOVI information model provides descriptors for resources and services, but it lacks of support for slice specification. Hence, it could not be adopted in the NECOS context. However, some of its classes, *e.g.,* the *Location* and *Lifetime* are useful and can be exploited by the NECOS information model to indicate and satisfy geo-location constrains, and specify time-related requirements for the creation and decommission of a slice. Limitations regarding the slice description and attributes to express service elements also exist in the 4WARD information model. The 4WARD model is primarily designed to specify objects and attributes of the infrastructure and the virtual network resources. The most suitable information model for network slicing is the COMS model. It provides a set of useful slice-specific attributes for NECOS, such as the starting and ending time of a service, the slice lifecycle status, as well as attributes related to slice requirements, *e.g.,* reliability levels, throughput threshold, latency or jitter agreement. The MANO information model also provides useful features with the EPA support being the most notable. In particular, the notion of EPA hides the heterogeneity among providers and exposes certain infrastructure features to the tenant. Host, hypervisor, VIM, vSwitch, interface, and service-end are some of the EPA attributes incorporated in the NECOS information model. Overall, the NECOS model incorporates features from the 4WARD, COMS and MANO models.

Along with the provisional information model described in the following section, cloud APIs are further defined exploiting the infrastructure and network slicing features exposed by the information model. Regarding the APIs, the NECOS takes advantage of the 5GEx proposal, which defines three main network interfaces: (i) between the customer and the multi-domain orchestrator for service exposure (B2C), (ii) for multi-domain orchestrators' interaction (B2B) and (ii) for the interaction between the multi-domain orchestrator and the domain orchestrators within the same administrative domain. In accordance with the aforementioned scheme, NECOS has defined *Client-to-Cloud APIs* and a set of interfaces composing the *Cloud-to-Cloud APIs*, which will be enhanced with slice-related methods for slice provisioning and run-time management. An API for the deployment, scaling and monitoring of infrastructure resources has also been specified by the OGF, (*i.e.*, OCCI); however, this API does not support network slicing.

The NECOS Marketplace allows for dynamic resource discovery and matching across multiple infrastructure providers for the instantiation of slices upon tenant requests. This goes beyond other Marketplaces (*e.g.,* NOVA), which are limited to service deployments within a single provider. NECOS also addresses the potential heterogeneity of participating infrastructure providers (*e.g.,* core, edge clouds) via the combination of efficient resource discovery/matching mechanisms with an information model that can expose particular features of the infrastructure to the Slice Broker. The NECOS Marketplace further provides support for elasticity, as it encompasses horizontal and vertical scaling. Elasticity constitutes a salient feature of the NECOS Marketplace, which will be further pursued until the end of the project. Finally, the NECOS Marketplace employs a cost model which provides the means for resource pricing and the computation of cost-effective slice deployments for the tenant.

3 NECOS Information Model

This section presents the final version of the NECOS information model used for slice provisioning and run-time management. The main objectives of the information model are: (i) the detailed description of all infrastructure resources/elements and their properties, and (ii) the description of slice components and service elements that could potentially be deployed within slices. To meet these requirements, we introduce an information model for network slicing, which provides resource descriptions at different levels of abstraction.

Initially, we consider three abstraction levels which are equivalent to the *Slice Database*, the *Tenant* and the *Infrastructure Provider* viewpoints (see deliverable D3.1 or D3.2). In Section 3.1, we discuss a high-level representation of the whole model, which currently resembles the *Slice Database* view, since the latter component keeps track of all required information for the slice operation. In Section 3.2, we describe in detail the slice specification model, *i.e.,* the objects and properties of the information model required to specify/request slices and address certain slice components or service elements deployed within slices; this comprises the *Tenant*'s view of the model. In Section 3.3, we provide a detailed specification of the model's objects and properties for the infrastructure description. This is essentially the *Infrastructure Provider*'s view, which is detailed, and its limited representation is communicated through the *Slice Agent* and *Slice Controller* components to the NECOS architecture (see D3.2), due to competition purposes, *i.e., Infrastructure Providers* are not willing to share detailed information about their infrastructure. In Section 3.4, we consider a web load-balancing slice example and highlight key features of the NECOS information model using simple YAML descriptions.

In the following, we discuss in further detail the three views of the model.

3.1 Information model overview

A model instantiation very close to the unified NECOS information model is at the heart of the NECOS platform, the *Slice Database* which is tightly coupled with the *Resource Orchestrator*. Such database keeps track of all information on the deployed and operating slices. This model representation bridges the network slice specification defined from the *Tenant* with the resource representation defined from the involved data-centre and WAN providers. As discussed above, the *Slice Database* stores a limited view of the infrastructure that provides resources for the slice. The term *"limited"* reflects the limited information disclosure exercised by infrastructure providers on third parties. A high-level representation of the unified NECOS information model is shown in Figure 4.

In particular, a slice description contains a Network Slice Specification and a Slice Infrastructure Description. The former is generated based on a service graph, which consists of service functions (*fn*) and service links. Service functions are further decomposed to service elements. The service functions represent the specifications for functional entities to be instantiated and the service elements their actual instantiation.

The *Slice Infrastructure Description* contains elements that reflect the slice infrastructure graph, *i.e.,* DC and Network slice parts, which may be deployed on top of separate cloud or network domains. These parts are linked to the service-elements that they host, as depicted in Figure 4. For instance, a "dc-slice-part" contains fields reporting provider-specific information, *e.g.,* which slice provider hosts the specific part, a VIM or a DC Controller. In addition, each slice part has one or more references to the service elements it will host. Obviously such information will be available after the successful completion of the resource discovery process. Examples of YAML messages regarding the Slice Infrastructure Description are provided in Section 4.

In the following subsections, we elaborate on the two primary aspects of the NECOS information model, the Network Slice Specification and the Infrastructure Description. As an outcome of our extensive literature research (*i.e.,* Section 2 and Appendix), the NECOS model is influenced by the MANO, NOVI, and COMS information models.

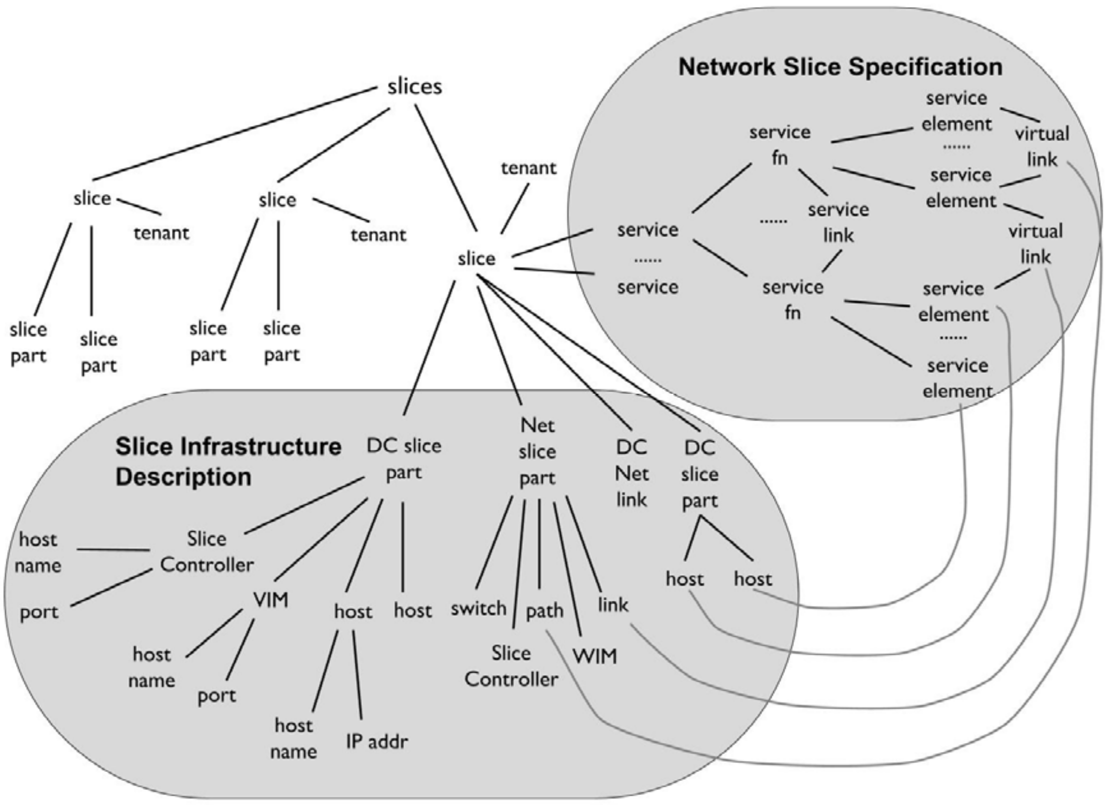


Figure 4. Overview of the NECOS information model.

3.2 Network slice specification

As NECOS deals with the provisioning, configuration, and run-time management of network slices, the NECOS information model needs to encompass all information required by the *Tenant* in order to specify slice and address slice components as well as service functions deployed within slices. Essentially, the information model will be used in different occasions by the *Tenant*, *e.g.*, when a *Tenant* submits a request for slice creation, when a *Tenant* wishes to submit a request on an existing slice for the modification of certain slice components or the scaling of the slice.

One of the challenges posed in terms of slice specification is that a slice may represent different views and/or may serve different purposes. For example, a slice may simply correspond to a subset of the physical infrastructure, essentially comprising a set of infrastructure elements, such as cloud servers. In this case, a *Tenant* will be granted with such slice and may later decide which services or applications he/she wishes to deploy. However, it should be also possible for a NECOS system to provision and manage service-tailored slices, *i.e.*, slice specifications that contain a set of service functions, such as VNFs. In the NECOS approach, this diversity in terms of slice request will be dealt with the translation of service demands into resource demands, facilitating resource assignment and allocation for slice creation. Nevertheless, in terms of slice specification, NECOS introduces a versatile information model that meets the requirements of all these aforementioned slicing levels, *i.e.*, by completing different sets of attributes.

The slice viewpoint of the NECOS information model is illustrated in Figure 5. The model associates slices with services, which are in turn, associated with services functions and corresponding service elements (*i.e.*, instantiated service functions). The information model further provides specifications of the VIM (which will be instantiated on-demand in *Mode 0*). This allows the tenant to express preferences for the VIM (*e.g.*, request the instantiation of OpenStack). This is facilitated by the notion of Extended Platform Awareness (EPA), which will be explained in more detail below.

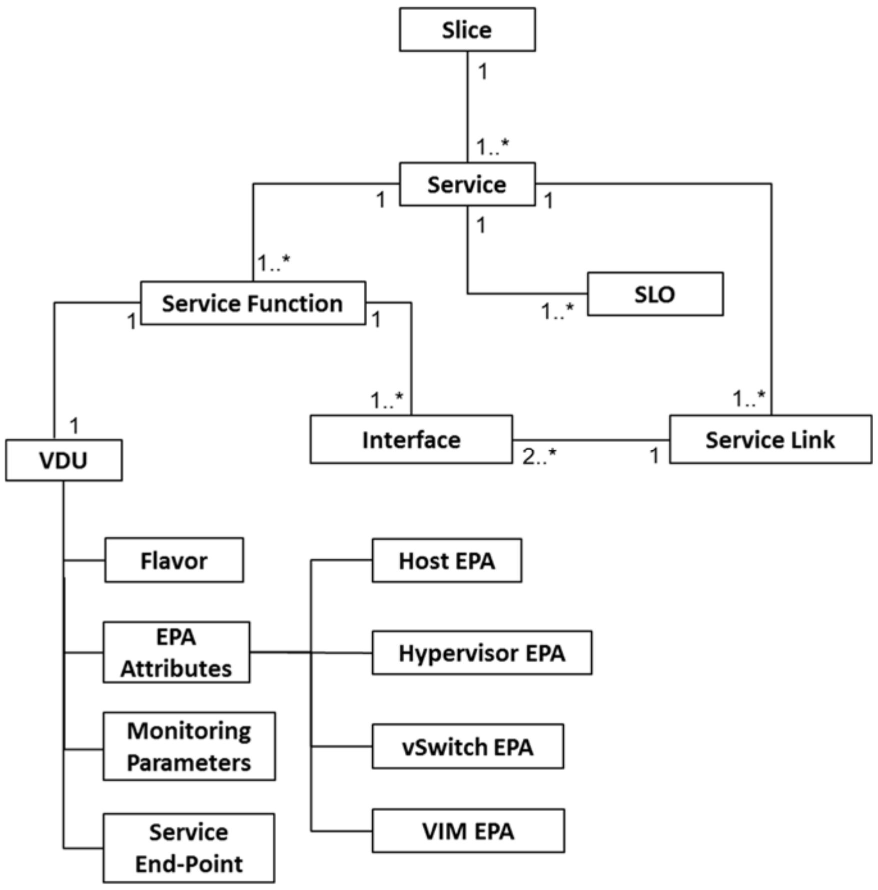


Figure 5. Slice specification with the NECOS information model.

In the following, we elaborate on the objects and attributes of the information model for the slice specification:

Slice: The ***Slice*** object provides a general description of the slice, which is further associated with services, as shown in Figure 5. This object includes the following attributes:

- Slice ID
- The service deployed within the slice
- General service requirements, for example: cost model, slice lifetime (*e.g.*, start-time and end-time), geographical or other slice constraints (e.g., maximum number of slice parts), etc.

Service: The ***Service*** object provides a general description of a service, which is associated with ***Service Functions***, ***Service Links***, as well as ***Service Level Objectives (SLO)***. This object includes the following attributes:

- Service ID
- Service description
- List of service functions
- List of service links

Service Function: The ***Service Function*** object is a descriptor for a service function, which is associated with ***Virtual Deployment Units (VDU)*** and interfaces. This object includes the following attributes:

- Service function ID

- Location preference, which can be specified in the form of coordinates or with the name of a location (e.g., *city*)
- Distance tolerance from preferred location
- Number of interfaces
- Placement group (*e.g.*, isolation, co-location), which can be used to enforce the co-location or isolation among a group of service functions

Service link: This object describes a link and is associated with interfaces. The object has the following properties:

- Service link ID
- Bandwidth, which specifies the amount of required bandwidth
- Delay, which specifies an upper bound on delay for this link
- Jitter, which specifies an upper bound on jitter for this link

SLO: This object provides a descriptor for SLO, which is associated with the ***Service***. We discuss in more detail the SLO description in Section 3.2.1.

VDU: The ***VDU*** object binds a service function with resource requirements, a range of EPA attributes, and monitoring parameters, providing great flexibility in the specification and monitoring of service functions. This object includes the following attributes:

- VDU ID
- VDU name
- Number of service function instances
- Flavor
- Software image

Flavor: The ***Flavor*** object specifies resource demands for services functions and has the following attributes:

- Flavor ID
- Number of CPUs
- Amount of main memory
- Amount of storage space

EPA: Inline with ETSI NFV MANO, we employ the notion of EPA to assist tenants in expressing preferences for the instantiation of service functions and VIMs. In a similar manner, we have further extended the application of EPA to virtual switches and hypervisors.

Host EPA: This object specifies the following EPA attributes for the host at which a service function will be deployed:

- CPU Model (Required / Preferred), *e.g.*, Westmere, Sandybridge
- CPU Architecture (Required / Preferred), *e.g.*, x86, x86_64, i686, ARM, etc.
- CPU Instruction sets (Required / Preferred), such as AES
- Acceleration (Required / Preferred)
- Acceleration technique, *e.g.*, DPDK, Netmap, etc.

Hypervisor EPA: This object specifies the following EPA attributes for the hypervisor deployed in a cloud domain:

- Hypervisor, such as Xen or KVM
- Version, which indicates a preferred version of the requested hypervisor

vSwitch EPA: This object specifies the following EPA attributes for virtual switches that can be deployed in the slice:

- Acceleration (Required / Preferred)
- Acceleration technique
- Hardware Offloading (Required / Preferred)

VIM EPA: This object specifies the following EPA attributes for the VIM, which will be set up to manage and control the slice:

- VIM, such as Openstack, OpenVIM, etc.
- Version, which indicates a preferred version of the requested VIM
- Dedicated VIM (Required / Preferred)

Service end-point: This object specifies an end-point for services, augmenting the binding of service with other applications or services. This object includes the following attributes:

- Service end-point name
- Type, *i.e.,* ingress or egress
- Interface
- Application
- Port number
- Protocol, *i.e.*, TCP or UDP

Monitoring parameters: This object contains a wide range of parameters for monitoring the performance and reliability of service functions. Examples of such parameters include counters for CPU, memory, network bandwidth or disk I/O, as well as the monitoring granularity. Some of the parameters may be associated with SLOs, augmenting the client in assessing the achievable performance of a cloud service (see Section 3.2.1). Figure 6 shows an example of monitoring specification, including (among other parameters) the monitoring tool, a list of monitoring metrics and the granularity for each one of the metrics.

```
monitoring-parameters:
       tool: prometheus
       measurements-db-ip: undefined
       measurements-db-port: undefined
       type: <container/host/all>
       metrics:
          - metric:
              name: PERCENT_CPU_UTILIZATION
              granularity-secs: 10
          - metric:
              name: MEGABYTES_MEMORY_UTILIZATION
              granularity-secs: 10
          - metric:
              name: TOTAL_BYTES_DISK_IN
              granularity-secs: 10
```

```
- metric:
    name: TOTAL_BYTES_DISK_OUT
    granularity-secs: 10
- metric:
    name: TOTAL_BYTES_NET_RX
    granularity-secs: 10
- metric:
    name: TOTAL_BYTES_NET_TX
    granularity-secs: 10
```

Figure 6. Example of monitoring specification.

3.2.1 Description of Service Level Objectives

We hereby discuss the modelling of SLOs, as part of the NECOS information model. While NECOS assumes that its management tasks are driven by the existence of SLAs between the tenants and the providers, as well as between the different providers, it has left the details of such SLA specification out of the scope of the project. However, in order to try the different elasticity concepts developed in the project (elasticity mechanisms will be documented in D5.2), we have decided to describe in more detail a particular section of common SLAs, *i.e.,* the SLO, leaving outside legal and economic aspects regarding SLA violation.

In the cloud context, there are a handful of SLA specification languages, most of them can be seen in [MAAROUF2016]. It easy to extend them to consider the slicing and networking aspects introduced by NECOS. Among those languages, because its compatibility with the concepts used in NECOS and its simplicity, we have decided to use the SLO specification language named SLO-ML which is presented in [ELHABBASH2019].

As mentioned in [ELHABBASH2019], SLO-ML adopts a JSON syntax for representing SLOs. Its elements are: Names, Value types, Units, and Operators. A unique keyword name is used to refer to each SLO. The keywords should be self-explanatory, making it simple for developers to understand, as seen in the example depicted in Figure 7.

```
aws_compute_web: [
   { name: ‘response time’
     unit: ‘ms’
     value: ‘5-10’
     operator: ‘in’
   }
   { name: availability
     unit: ‘’
     value: ‘0.999’
     operator: ‘>=\’
   }
]
```

Figure 7. SLO description example.

SLO-ML supports three types of SLO values: scalar, interval, and categorical:

- The scalar type is used to specify a numerical value (*e.g.*, availability = 0.9999).
- The interval type is used to specify an upper- and lower-bound of SLO value (*e.g.,* response time between 5ms and 10ms).
- Categorical types provide a higher level of abstraction for SLO value specification, allowing customers to specify a category (*e.g.,* low, medium, high) instead of specific values or a predefined range, relieving customers from specifying an exact value in case they are not certain.

For example, for memory-intensive application, a customer can specify the category ‘high’ for the Memory Size SLO. SLO-ML uses a set of keywords that specify the units of measurement of each SLO. For example, the Migration Time SLO is specified using the hour unit. In addition, SLO-ML contains rules for unit-to-unit conversion between units of the same kind. SLO-ML defines a set of operators to specify relational SLO values. This set includes: less than, less than or equal, greater than, greater than or equal, equal, and in. For instance, it can be used to indicate that response time should be in the interval [5ms,10ms].

3.2.2 Description of Policies

Since its advent, Policy-based Management (PBM) has left behind its more or less distended expectations to get into being just another standard tool in the day-to-day network and cloud operations. All major cloud and network softwarisation tools include policy-based management capabilities to some extent. NECOS is designed to deal with these features and needs to, at least, model those policies as just another feature of the underlying technologies.

In a multi-provider environment, there is also the need to address the problem of deciding when two resources that are managed by different entities and that respond to different policies, can be used together. Moreover, once those resources are connected into a single network cloud slice, NECOS needs to have a view of how all the local and global policies interoperate, those granting or forbidding access and those driving the behavior of a slice and the services deployed on it. In addition, the collaboration between different providers enabled by NECOS can be regulated by a contract (implemented as a set of policies) negotiated between the different entities and applied to control their interoperation.

To cope with these needs, we have designed a basic policy model, an adaptation of the NOVI Policy Model [NOVI2015], that can represent the most common policies. We can model access control policies, understood in a broad sense, not only as security policies, and behavior-controlling policies, such as those that command an action after some slice-related event fulfill some conditions; these are called event-condition-action (ECA) policies.

In this respect, NECOS information model supports the following policies:

- **Event-condition-action policies** that enforce control and management actions upon certain events within the managed environment.

- **Role-based-access control policies**, used to assign users to specific roles, and where different permissions/usage priorities on virtualized resources are granted to each role.

The main entities of the NECOS Policy Submodel are the following:

- **ManagedEntity.** It is the entity that the Policy Manager is able to manage, interpreting this management in a broad sense. It may be, for example, an entity being passively monitored or an entity that acts just as the Policy Manager indicates. It includes physical and virtual resources, end services and slice services within a NECOS federation;

- **User.** This may be a user of NECOS or a part of a NECOS federation acting as a user of other.

- **Action.** This comprises an action that can be performed by a ManagedEntity;

- **Event.** This is an Event that can be caught by the Policy Service to trigger the proper actions;

- **Condition.** This comprises a condition that needs to be checked before triggering the actions;

- **EntityProperty.** This is a property of a ManagedEntity. These properties can be used as conditions for Policies.

Furthermore, A *ManagedEntity* comprises the following subclasses:

- **ManagementDomain** is a Domain that may contain other Management-Domains and ManagedEntities. It gives the ability to manage the entities in a common way. It is also a subclass of ManagedEntity to give the ability to be managed by itself;
- **Policy** is an abstract class to define policy rules that will be used to manage the ManagedEntities;
- **EPA Resource** is the resource on which the Policies are applied;
- **Slice** is the slice service on which the Policies are applied;
- **Event** is the event as it is stored in the Policy Engine, typically collected by the Monitoring capabilities of IMA, *e.g.*, it may be a threshold crossing, a topology change, a SLO violation, etc;
- **Role** is the role of a NECOS user.

3.3 Infrastructure description

The physical infrastructure spans datacenters (that provide computing resources) and wide-area (or transport) networks that provide connectivity between the datacenters from which resources will be allocated for the slice instantiation. The NECOS information model aims at capturing the main resource and network elements available in datacenter and transport networks, such as servers, routers, switches, controllers, links, etc. Each object in the information model is associated with a set of properties that represent certain attributes for the infrastructure element. The range of properties for each object is certainly not exhaustive but can be easily extended with additional properties, if needed.

In the following, we present the main objects that are supported by our information model, including their main properties.

Infrastructure: The ***Infrastructure*** object provides a general description of the whole infrastructure, on top of which, network slices will be instantiated. This object of the information model encompasses the following attributes:

- Infrastructure ID
- List of domains comprising the infrastructure

Domain: The ***Domain*** object provides a general description of an infrastructure domain, which, in the case of NECOS, may be either a datacenter network (which corresponds to a traditional cloud), an edge cloud, or a WAN that provides connectivity among different datacenters. This object has the following attributes:

- Domain ID
- Provider, *i.e.,* the infrastructure provider for this domain
- Type, *i.e.,* datacenter for traditional clouds, (mobile) edge cloud, or WAN (see the respective telco cloud and MEC use cases in the deliverable D2.1)
- List of the hosts in the corresponding network domain
- List of the network elements in the network domain
- List of the links available in the network domain

Network Element: The ***Network Element*** object provides an abstraction of network elements, such as routers, switches, and Wi-Fi access points (AP). A network element includes the following attributes:

- Network element ID
- Availability
- Type, *i.e.*, the type of network element, such as router, switch, or AP.
- Number of ports
- Forwarding Information Base (FIB) size
- Cost

Router: This object describes a router and includes the following attributes:

- Router ID
- Role, *i.e.*, edge or core router
- Packet types that can be processed, such as IP, MPLS, Ethernet
- Monitoring parameters for the router

Switch: This object describes a datacenter network switch and includes the following attributes:

- Switch ID
- Role, *e.g.*, Top-of-the-Rack, Aggregation, or Core switch
- Packet types that can be processed, such as IP, MPLS, Ethernet
- Monitoring parameters for the switch

Access Point: This object describes a Wi-Fi access point and includes the following attributes:

- Access Point ID
- Availability
- MAC, *i.e.*, MAC specifications supported, such as 802.11n
- Monitoring parameters for the AP

Host: This object describes hosts and encompasses the following attributes:

- Host ID
- Hostname
- Availability
- Location
- CPU, which contains a pointer to a separate object, namely CPU
- Memory, *i.e.*, the amount of available main memory
- Storage, *i.e.*, the amount of available storage capacity
- Number of ports
- Monitoring parameters for the host
- Other service-specific host capabilities, e.g., energy-measurement hardware, SAS disks optimized for storage nodes, etc.
- Cost

CPU: This is a dedicated object for CPU-related specifications. The *CPU* object is associated with the *Host* object and contains the following attributes:

- Cycles, which specifies the number of available CPU cycles per core
- Number of cores
- Model, *e.g.*, Westmere, Sandybridge
- Architecture, *e.g.*, x86, x86_64, i686, ARM
- Instruction set, *e.g.*, AES

Controller: This object is associated with hosts, providing additional properties for hosts that serve as DC or network controllers. The ***Controller*** object contains the following attributes:

- Controller ID
- Role, *i.e.,* DC or network controller
- Configuration protocol, *i.e.,* the protocol used to configure the DC or the network (*e.g.,* SNMP, YANG, OpenFlow)
- Configuration IP address

Link: This object describes network links with the following attributes:

- Link ID
- Availability
- Type of the link, *e.g.,* point-to-point, point-to-multipoint
- Capacity
- Delay
- Jitter
- Cost

Port: This object describes network ports and include following attributes:

- Port ID
- Availability
- Capacity
- List of the queues that may have been configured in the port, in the case of hardware multi-queuing (e.g., SR-IOV)
- IP address
- MAC address

Queue: This object describes queues in network ports that could be potentially configured, when there is support for hardware multi-queuing. The ***Queue*** object has the following attributes:

- Queue ID
- Availability
- Capacity

Path: This object describes network paths that encompass a set of links. The *Path* object has the following attributes:

- Path ID
- Availability
- List of links that comprise the network path
- Capacity, which expresses the overall capacity of the path and corresponds to the minimum capacity of all links that comprise the path
- Delay, which expresses the total delay incurred along the path
- Jitter, which expresses the overall jitter incurred along the path
- Disjoint links, which requires that all links comprising the path are disjoint

The UML diagram in Figure 8 illustrates the objects of the information model for the infrastructure description, as well as the relations between these objects. We note that some of the objects have been omitted in the UML diagram for clarity. One such object is the Wi-Fi access point, which is connected to the ***Network Element*** object in a similar way with the other respective elements (*i.e.*, router, switch). This UML model can be easily implemented through more specific description languages or schemas, such as XML, RDF, and YAML.

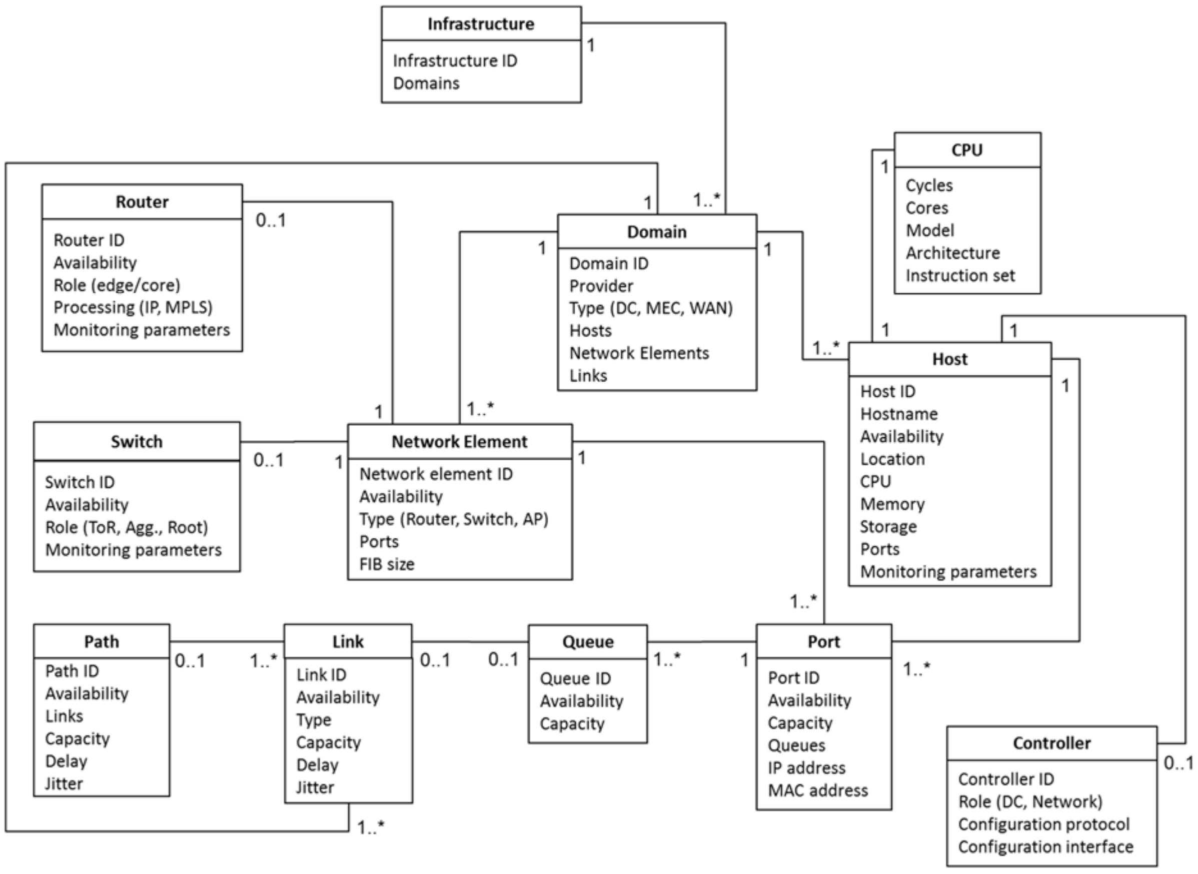


Figure 8. Infrastructure description with the NECOS information model.

3.4 YAML Description Example

This section presents a YAML example based on the NECOS information model. In particular, we consider a slice that hosts an *advanced* touristic content delivery service (CDN) that consists of o a core cloud "central" content delivery service (CDN) and a number of locally deployed edge cloud lightweight CDNs, with the aim to bring location-specific content closer to the end-users of the service. The main assumption guiding the design of the service is that users (tourists) located in high profile touristic areas are more likely to request information and video related to that specific area, thus served locally by edge cloud services. In more detail, the service consists of:

- A central advanced content service, hosting a number of web pages and video streaming services related to *touristic attractions around the world*, *i.e.*, this service hosts the entire content, as well as a load balancing service able to direct user requests to the most appropriate server (edge or cloud) based on user location and the content requested. This main service is referred to as "**core_vm**" hereafter.

- A number of lightweight web and video servers hosting touristic location-specific content, which are geographically distributed, in predetermined areas around the world. These services are named "**content_server_<country>**" hereafter, where <country> denotes the geographic location where the edge cloud service is located. For instance, "**content-server-spain**" refers to a content server in Spain.

- A number of benchmarking tools (*load testing tool-ltt*), which the tenant wishes to deploy in order to perform periodic evaluations of QoS. Similar to the above, these services are named "**ltt_<country>**", *e.g.,* "**ltt_spain**" is the name of a service located in Spain.

Communication among the above are necessary, *e.g.,* the load testing tool has to be able to contact the **core-vm** service to (i) discover the "best" server to address its request for content or (ii) to obtain content in the case that the "near-by" server cannot serve the latter, as well as be able to address a request to the closely located content server. Finally, the **core-vm** service has to be able to contact the edge cloud content services in order to change the content they serve, possibly based on populartity information.

Figure 9 presents an overview of the different services and their connectivity in the CDN. For brevity, we only depict the Core Cloud service and services associated with one edge cloud (in Spain). In the figure, boxes titled "Service", *e.g.,* the "core_vm" main content delivery service, represent the services of the CDN and for each such service interfaces are defined, as for instance the "core-eth1" and "core-eth0" interfaces for the aforementioned service. Interfaces act as ends points for "Service-links" depicted as boxes titled with the same keyword, *e.g.,* the "ltsp-to-core" service link connects the core content server with the load testing tool service hosted in the edge cloud server in Spain. Obviously, edge cloud services in different areas have a similar structure.

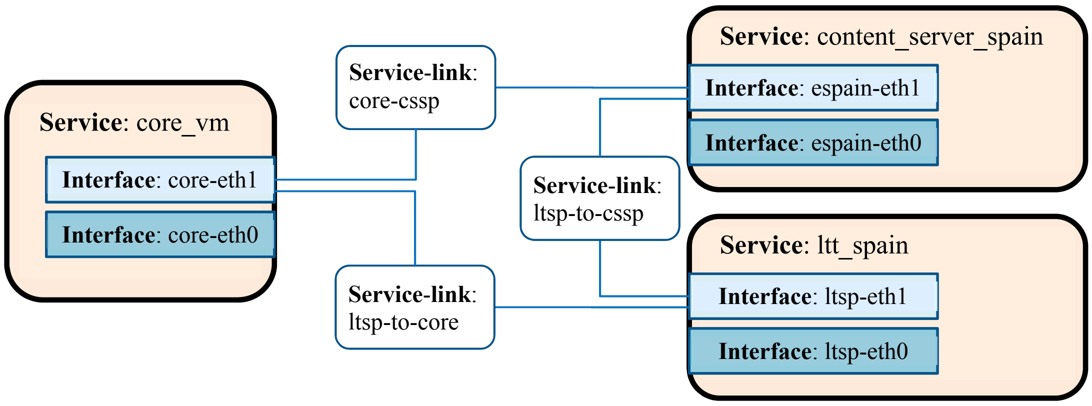


Figure 9. Touristic CDN diagram of services, service-links and interfaces.

In the following, a representation of the service graph described above is provided using the NECOS information model. The section of the YAML specification first provides general information regarding the slice (Figure 10), including geo-location slice constraints ***("location: [EUROPE, AMERICA]")***, the number of the requested DC and Network slice parts, as well as slice requirements such as elasticity. For reasons of clarity, some sections of the YAML description have been omitted.

```
slice-constraints:
         geographic:
            continent: [EUROPE, AMERICA] #--shmeiwsh
            country:
         dc-slice-parts: 4  # Could be a constraint on values.
         net-slice-parts: 3

      slice-requirements:
         elasticity: false
         reliability:
            description: reliability level
            enabled: true
            value: none   # {path-backup, logical-backup, physical-backup}

      slice-lifecycle:
         description: lifecycle status
         status: constuction    # {modification, activation, deletion}

      cost:
         dc-model:
            model: COST_PER_PHYSICAL_MACHINE_PER_HOUR
            value-euros: {lower_than_equal: 10}
         net-model:
            model: COST_PER_LINK_PER_HOUR
            value-euros: {lower_than_equal: 50}

      slice-timeframe:
         service-start-time: {100918: 10 pm}
         service-stop-time: {101018: 10 pm}
      # at least one slice component and one VDU should be defined
      service:
         - service-function:
             …
         - service-function:
             …
         - service-link:
           …
```

Figure 10. YAML top-level descriptions.

The top-level slice description contains a section regarding the cost for the DC and WAN slice parts, and the time-frame for which the slice is requested (***“slice-timeframe:”***). The service section of the YAML description includes service-elements' information, as for example, the “core_vm” service element specification is depicted in Figure 11. The service function specification contains all the necessary information for the NECOS system to create the slice hosting the service. For example, the specification contains the number of instances of the “core_vm” service element, (***instance-count: 1***), and the sections include EPA attributes. More complicated specifications are supported as those were described in D4.1.

```
    service:
         - service-function:
              # defining load balancer VDU
              service-element-type: vdu
              vdu:
                 id: core_vm
                 name: core_vm
                 description: load balancer and content services
                 instance-count: 1
                 hosting: DEDICATED
                 slice-part-count: 1
                 vdu-image: undefined
                 epa-attributes:
                    host-epa:

                    hypervisor-epa: ...
                    VIM-epa: ...

                    vswitch-epa: ...
                    ...
```

```
interfaces:
monitoring-parameters: ...
```

Figure 11. YAML specification of a service function.

For the specific service element, the associated EPA attributes are shown in Figure 12, where all resource demands and constraints are described.

```
host-epa:
  cpu-model: PREFER_CORE2DUO
   cpu-arch: PREFER_X86_64
   cpu-vendor: PREFER_INTEL
   cpu-number: 2
   storage-gb: 30
   memory-mb: 8192
   host-count: 1
   max-host-count: undefined
   os-properties:
      # host Operating System image properties
      architecture: x86_64
      type: linux
      distribution: ubuntu
      version: 16.04
   image-type: XEN
   host-image: ubuntu_linux_16.04_xen

hypervisor-epa:
   type: XEN
   version: '4.5'

VIM-epa:
   type: XEN-SERVER
   version: '7.5'
   vim-shared: true
   vim-federated: false
   vim-ref: undefined

vswitch-epa:
   type: openvswitch
   accellaration: PREFERRED
   offload: PREFERRED
```

Figure 12. EPA attributes specification.

The specification of the service function interfaces is defined by a corresponding YAML key, as shown in Figure 13. It should be noted that we differentiate between service external and service internal interfaces: the former are end-points for users to access the service offered by the slice externally, whereas the latter are interfaces for connecting to other service elements. Thus, the internal interfaces are associated with service links, as discussed below.

```
    # defining web-server cluster's interfaces
interfaces:
    - service-external-interface:
         name: core-eth0
         virtual-interface:
             internal-name: eth0
             type: VIRTIO
             bandwidth: '0'
             vcpi: '0000:00:0a.0'
             ip: undefined
    - service-internal-interface:
         name: core-eth1
         virtual-interface:
             internal-name: eth1
             type: VIRTIO
             bandwidth: '0'
             vcpi: '0000:00:0b.0'
             ip: undefined
    - service-internal-interface:
         …
```

Figure 13. Service function interface description.

In the specific service, for instance, the "core_vm" service element is linked to the edge cloud content service. This leads to the service-link YAML description depicted in Figure 14. A service link has ***"link-end-references"*** (***link-end-ref: espain-eth1 and link-end-ref: core-eth1***), where the values are internal interface IDs of service element parts. For instance, the service link between the core and edge cloud services (depicted in Figure 9), is given in Figure 14. The rest of the fields contain information necessary to allocate appropriate network resources for the service link.

```
- service-link:
     service-element-type: link
     link:
        name: ltsp-to-core
        type: MULTIPLEXED
        ends:
           - link-end-ref: espain-eth1
           - link-end-ref: core-eth1
        requirements:
           bandwidth-GB: 1
        constraints:
           hops: {lower_than_equal: 2}
        reservation-protocol: undefined
```

Figure 14. Service link description.

One or more service links will be mapped to a network-slice part, as it will be demonstrated in the sections that follow. This concludes this example for service specification. The following section presents the slice infrastructure description.

3.4.1 Slice Infrastructure Description

In NECOS, the tenant is allowed not only to provide the service specification, *i.e.*, service elements and service links as those described in the previous section, but also provide specifications regarding the desired slice topology, *i.e.*, a mapping of the service elements and service links to slice parts (Figure 4). This ability provides the tenant with additional control on the deployment of the service, which in some cases can be crucial. For instance, in the touristic content service used as an example in the previous section, edge cloud services geographic location is of crucial importance, since the tenant will most definitely require that those are located near high-profile touristic areas in order to maximize the benefits of the CDN model. Similar demands can arise in a number of use cases; for instance, entertainment content providers might request a slice part near an event location, such a football stadium hosting a popular game, or even an event of even higher importance, such as Olympic games.

In our working example, assume that the tenant wishes that both the edge cloud content-server and load-testing tool are located in the same slice part, deployed in Spain (and thus the name), whereas the core cloud server located in Brazil (Figure 15).

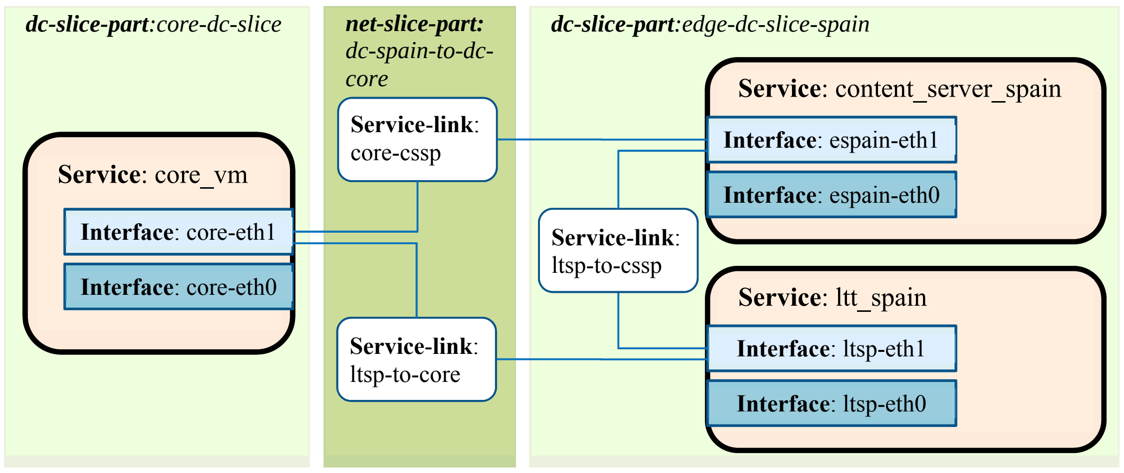


Figure 15. Slice Infrastructure for the Touristic CDN Service, showing the different (dc|net)-slice-parts that contain the services and the service-links.

This kind of information can be represented in the "slice" part of the YAML document describing user requirements. Consider for instance, the specification of the slice part that hosts the "core_vm" service, depicted in Figure 16. The slice part can carry a number of slice-constraints, including for example that of the geographic location. The reader should note that this geographic constraint, if it exists overrides the one found in the service specification mentioned earlier. Obviously, not all information related to the slice part can be defined at this stage: details regarding the provider's dc-controller will be filled in later (annotated as **undefined**) during the resource discovery phase.

```
slice:
   …
   - dc-slice-part:
        name: core-dc-slice
        slice-constraints:
           geographic:
              continent: AMERICA
              country: Brazil
        dc-slice-controller:
           dc-slice-provider: undefined
           ip: undefined
           port: undefined
        VIM: undefined
        vdus:
           - dc-vdu:
                id: core_vm
                name: core_vm
                description: load balancer and content services
                host-count-in-dc: 1
                max-host-count-in-dc: 1

        vim:
           name: core-xen-vim
           type: XEN-SERVER
           on-demand: true
           host-count: 1
           max-host-count: 1
           image: 'core_vm'
           hypervisor:
              type: XEN
              version: '4.5'

        dc-slice-point-of-presence:
            pop-name: undefined
            ip: undefined
            router-type: undefined
            reservation-protocol: undefined
            requirements:
                bandwidth: 10 Mb
```

Figure 16. Specification of the dc-slice-part hosting the "core_vm" service.

Such a slice part is associated with services in the VDUs section that contains a dc-vdu entry for each service element that it will host. For instance, in the case of the edge cloud server in Spain, the corresponding section will include two dc-vdu's.

```
     dc-slice-part:
        name: edge-dc-slice-spain
        slice-constraints:
           geographic:
              continent: EUROPE
              country: Spain
        dc-slice-controller:
           dc-slice-provider: undefined
           ip: undefined
           port: undefined

        VIM: undefined
        vdus:
           - dc-vdu:
                id: content_servers_spain
                name: content_servers_spain
                host-count-in-dc: 1
                …
           - dc-vdu:
```

```
id: ltt_spain
name: load_testing_tool_spain
description: load testing tool for CDN deployment
host-count-in-dc: 1
```

Figure 17. DC-slice-part section in the YAML File.

The dc-slice-part do not contain any information on the resources; the latter is retrieved via the links to the corresponding service elements. This decision not to duplicate that information was taken for reasons of representational economy. It is interesting to point that in this case the service link (*ltsp-to-cssp* in Figure 15) connecting the two service elements must be handled by the DC provider who will host the part, ensuring that any constraints on the connection are met.

On the other hand, for the service-links between the edge and the core cloud (ltsp-to-core and core-cssp) a network slice part (net-slice-part: dc-spain-to-dc-core in Figure 15) is to be allocated, as shown below (Figure 18).

```
- net-slice-part:
          name: dc-spain-to-dc-core
          wan-slice-controller:
             wan-slice-provider: undefined
             ip: undefined
             port: undefined
          WIM: undefined

          links:
             - dc-part1: core-dc-slice
             - dc-part2: edge-dc-slice-spain
          type: interaction
          accommodates:
             - service-element: core-cssp
             - service-element: ltsp-to-core
          link-ends:
             link-end1-ip: undefined
             link-end2-ip: undefined
```

Figure 18. Net-slice-part section in the YAML File.

Obviously, this is one of the possible cases. The tenant might wish, for some reason, to allocate the two links to different providers, which means that there will be two net-slice-part entries in the YAML document describing the desired slice.

Finally, in the extreme case that the tenant wishes to request only infrastructure resources, the same representation can be followed, using "dummy" services only to state "epa-attributes" describing resource characteristics in each slice part and network connectivity constraints that the tenant wishes to include in the request.

In the following section, we provide further details about the requirements of the different parts in order to exemplify the ***Resource Discovery*** phase and better illustrate the information exchange that takes place.

4 Marketplace

The resource discovery framework, as described in the NECOS architecture work package (*i.e.,* WP3), is responsible for locating the appropriate resources that compose a slice, *i.e.,* slice components that correspond to service functions and service links, according to the information model. A **Partially Defined Template (PDT)** message defines the general slice requirements and acts as the input to the resource discovery framework. This message is created by the *Slice Specification Processor* and passed to the *Slice Builder*. The *Slice Broker* is responsible to locate resources from both DC and WAN providers to fulfil the slice requirements and prepare a corresponding response, namely a **Slice Resource Alternatives (SRA)** message. In practice, the SRA message annotates the PDT message with alternative slice component options.

This process involves the following three architectural, functional components:

- *Slice Builder*, responsible for forming an appropriate request to the Broker (*i.e.,* a PDT message), and selecting the most appropriate slice components, among alternatives returned by the Broker in the form of an SRA message.

- *Slice Broker*, which receives requests from the Builder and replies with alternative responses that fulfil the request, *i.e.,* creates and responds with an SRA message for each PDT message it receives.

- *Slice Agent*, which resides in the DC/WAN provider domain, replying to resource queries from the Slice Broker. The *Slice Agent* receives the slice component requirements, checks the local resource availability through communicating with its own *DC/WAN Controller* and responds with one or more resource options.

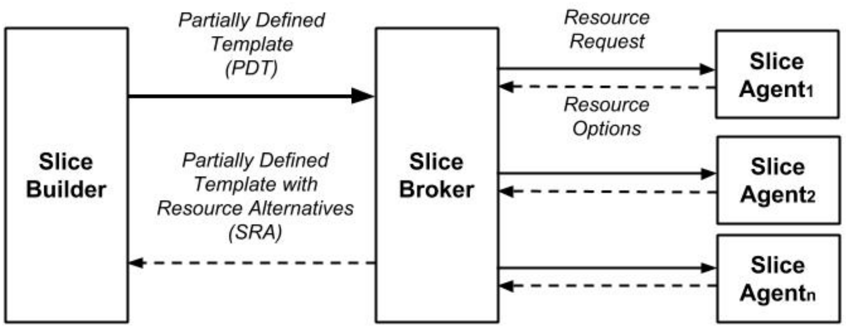


Figure 19. Overview of the resource discovery workflow.

Although the above description implies a query/answer model, in which data regarding availability of resources is dynamically collected for each request, a different model in which providers "push" information to the *Broker Agent* might as well be used, with minor modifications in the flow described below.

In the following, we highlight a **basic slice resource discovery workflow**. We assume that the builder has already prepared a PDT message, which includes the preferable number of slice components, their main resource requirements and the desirable connectivity among them. The basic steps of the workflow are as follows:

1. The *Builder* sends to the *Broker* a request in the form of a partially defined slice template.
2. The *Broker* proceeds with the incoming slice request processing through the following steps:

2.1. It decomposes the PDT message to simpler messages corresponding to slice parts, aggregating in each message all the information required for the agents to check resource availability
2.2. It queries the *DC Slice Agents* about the DC resources that are requested in the slice template.
2.3. Subsequently, the *Broker* processes the DC resource responses received from the *Slice Agents*. For sets of resource components that match the template, the broker identifies potential network resource providers, and sends queries according to the connectivity demands indicated in the template.
2.4. Finally, it collects available information, in the form of alternative resources for each template's slice component and conveys the response to the *Builder*.

3. The *Builder* receives the above information in the form of an SRA message and decides on the final slice specification, completing the slice specification template and instantiating slice components with the allocation of the respective resources.

In the following, we elaborate further on the form of messages exchanged between the components stated above.

4.1 Resource Discovery

4.1.1 Partially Defined Template

The creation of the Partially Defined Template is the work of the *Slice Specification Processor*, which is responsible for creating an initial template based on user service specification. Given that, as stated in the previous section, the tenant is allowed to define aspects of the slice topology, in some cases, the role of the Slice Specification Processor will be to complete the tenant's partial topology description with additional information.

In the simplest case, the template has a complete specification of each component (resource) required and only matching resources are returned in the responses. A more flexible setting involves a template component to be annotated with general slice or resource-specific requirements, so that resource providers can respond in a more flexible manner.

The PDT template must include both the desired slice topology, which encompasses the desired slice-parts, along with any resources' constraints on them and their connectivity (*i.e.*, the slice graph), and the service specification, since the latter contains the epa-attributes necessary for the discovery. For instance, below we present an abstract view of a simple exemplary template, represented in YAML (Figure 20):

```
sliced:
  id: TouristicCDN_sliced
  ...
  slice-constraints:
    geographic:
      continent: [EUROPE, AMERICA]
      ...
  slice-requirements:
    elasticity: false
```

```
      ...
   slice-lifecycle:
      ...
   cost:
      dc-model:
         model: COST_PER_PHYSICAL_MACHINE_PER_HOUR
         value-euros: {lower_than_equal: 10}
      net-model:
         ...
   slice-timeframe:
   service:
      - service-function:
           service-element-type: vdu
           vdu:
              id: core_vm
              name: core_vm
              ...
              epa-attributes:
                 host-epa:
                    cpu-memory-mb: {greater_than:2048}
                    cpu-architecture: ...
                    ...
              service-end-point:
              monitoring-parameters:
                 tool: custom
                 ...
           interfaces:
              - service-external-interface:
                   name: core-eth0
                   ...
              - service-internal-interface:
                   name: core-eth1
                   ...
      - service-function:
           service-element-type: vdu
           vdu:
              id: content_servers_spain
              ...
      - service-function:
        ...
        - service-link:
              service-element-type: link
              link:
                 name: core-cssp
                 ...
        - service-link:
              ...
        - ...
    slice:
        - dc-slice-part:
              name: core-dc-slice
              ...
        - dc-slice-part:
              name: edge-dc-slice-spain
              slice-contraints:
               vdus:
                 - dc-vdu:
                      id: content_servers_spain
                      …
                 - dc-vdu:
                      id: ltt_spain
        - dc-slice-part:
              ...
        - ...
        - net-slice-part:
              name: dc-spain-to-dc-core
              wan-slice-controller:
                 wan-slice-provider: undefined
                 ip: undefined
                 port: undefined
              WIM: undefined
              ...
        - net-slice-part:
              ...
        - net-slice-part:
              ...
        - ...
```

Figure 20. The structure of the PDT Message in YAML.

Figure 20 indicates that the slice topology is clearly reflected in the YAML message. The two dc-slice parts (*i.e.*, the ***core-dc-slice*** and ***edge-dc-slice-spain***) should be connected via a network link, represented by the ***dc-spain-to-dc-core*** inside the corresponding **net-slice-part**. We consider the former as nodes in the slice graph, while the latter corresponds to edges.

As stated, each part carries information regarding the desired characteristics that should be met when instantiating the component.

Values in the different fields of the YAML file can be either:

- ***ground*** numerical/string values, *i.e.*, integers or strings,
- ***undefined*** indicating that the value has to be filled with the appropriate information, once the Builder selects one of the candidate providers' offers returned by the *Broker*, or,
- expressed as ***relational constraints*** that the completed by the provider value must satisfy. For instance, a value ***{greater_than:2048}*** in the cpu-memory-mb field indicates that the desired host must have an amount of RAM greater that 2GB.

Undefined and constrained values justify the term "partially defined" of the PDT message, since they act as "variables" in the slice structure. As such, it contains sufficient information for the rest of the components to discover resources along with undefined/constrained fields to be completed later in the discovery process, as discussed in the section that follows.

4.1.2 Resource Queries

The *Broker* decomposes the PDT message it receives from the *Builder* and creates a different query for each slice component. Given the structure of the PDT message, such decomposition is performed easily, since each slice component corresponds to a different resource provider. The *Broker* has all the necessary information to form query messages that contain all the constraints/preferences/resources needed for the component request message. Thus, the Broker collects all information regarding the part, located in different sections of the YAML request file. For instance, such a message is depicted in Figure 21:

```
slice:
   …
   - dc-slice-part:
        name: core-dc-slice
        slice-constraints:
           geographic:
              continent: AMERICA
              country: Brazil
         cost:
           dc-model:
              model: COST_PER_PHYSICAL_MACHINE_PER_HOUR
              value-euros: {lower_than_equal: 10}
        dc-slice-controller:
           dc-slice-provider: undefined
           ip: undefined
           port: undefined
        VIM: undefined
        vdus:
           - dc-vdu:
                id: core_vm
                name: core_vm
                description: load balancer and content services
                host-count-in-dc: 1
                max-host-count-in-dc: 1
                epa-attributes:
                 ...
        vim:
           name: core-xen-vim
           type: XEN-SERVER
           on-demand: true
           host-count: 1
           max-host-count: 1
```

```
                image: 'core_vm'
                hypervisor:
                   type: XEN
                   version: '4.5'

             dc-slice-point-of-presence:
                 pop-name: undefined
                 ip: undefined
                 router-type: undefined
                 reservation-protocol: undefined
                 requirements:
                     bandwidth: 10 Mb
```

Figure 21. Broker to DC Slice agent query message.

As shown in Figure 21, the message contains a number of sections that need to be filled by the *Slice Agent* when responding positively. For instance, information regarding the dc-slice-controller and the point-of-presence need to be completed with its own information and the cost model section. Instantiated values act as constraints with respect to the slice part (*e.g.*, elasticity). Constraints regarding the specific hosts that will host the VDUs are described in section ***vdus***, as those are found in the epa-attributes section of the YAML file.

A similar message is sent to the *WAN providers* (Figure 22) that contains the necessary information for allocating the link between DC slice parts. Messages to WAN providers are sent after processing of the DC Providers' replies.

```
net-slice-part:
   # Same information as in the DC-above
   slice-constraints: …
   slice-requirements:
   slice-lifecycle: …
   cost: …
   slice-timeframe: …
   wan-slice-controller:
      wan-slice-provider: undefined
      ip: undefined
      port: undefined
   WIM: undefined
   links:
      - dc-part1:
           name: core-dc-slice
           dc-slice-point-of-presence:
              pop-name: defined-previous-step-by-broker
              ip: defined-previous-step-by-broker
              router-type: defined-previous-step-by-broker
              reservation-protocol: undefined
              requirements:
                    bandwidth: 1 GB
        - dc-part2:
                name: edge-dc-slice-spain
                …
    type: interaction
    constraints:
        bandwidth: 1 GB
    link-ends:
        link-end1-ip: undefined
        link-end2-ip: undefined
```

Figure 22. Link specification in YAML.

It should be noted that the same scheme can be alternatively implemented by sending complex queries to the *Slice Agents* in order to report availability on multiple components of the slice. The advantages of this alternative approach will be further investigated in the future.

4.1.3 SRA Messages

The *Broker* collects all alternative responses to the messages above, and sends the response to the Builder. Responses for each alternative slice-part (both dc-slice parts and network-slice parts) are, in fact, lists of alternative resources originating from the *Providers*' *Slice Agents*. Since each *Slice Agent* supplies references to the offered slice parts, along with cost and other information, the *Builder* is in position to select the configuration of the slice that best matches the client's needs.

This concludes the initial presentation of the slice resource discovery workflow. Next steps include the *Slice Builder* instantiating slice parts to resources offered by providers and finally passing this information to the *Slice Orchestrator* to complete the slice creation according to the deliverable D3.2. The following sections discuss two issues related to the resource discovery, namely, *resource matching* and the *cost model.*

4.2 Resource Matching

The role of resource matching is to map a request message to the available Provider's resources. However, such a mapping can rarely be *exact*, in the sense that resource characteristics as stated in the request message and those available in the resource provider side might not be an *exact match*. For instance, assume that the resource request states that **`storage-gb`** epa-attribute to be 4, meaning that a host with 4GB of disk space is required. If the provider has a host available with a storage capacity of 8GB, then it should (probably) answer positively to the request since its offer specification exceeds that of the request. We refer to this issue as the *resource coverage problem*, *i.e.*, finding those resources that meet or exceed (cover) the requirements stated in the request message.

The problem can be even more complex in the sense that some of the attributes defined in the request message demand different relational operators to ensure coverage. For instance, the attribute mentioned in the previous paragraph is "covered" by comparing available using the "greater_or_equal" relational operator (>=), while others, as for instance **`delay`** in a network resource need to be checked using the "less_than_or_equal" operator. Given the large number of attributes that have to be matched, the complex information model described in the previous section, and the diverse infrastructure representations that providers might use, the matching mechanism relies on the notion of *matching rules*, that have the following form:

matching_rule (RuleId, YamlPath, ProviderResourcePath, Operator)

where **RuleID** has the usual meaning, **YamlPath** is the attribute's path in the YAML resource request message, **ProviderResourcePath** is the corresponding Provider's infrastructure representation path to the matched characteristic and finally **Operator** is the binary relational operator used to compute coverage. For instance, if we consider that the Provider's Agent has a list of available resources stored in a YAML representation, an example of a rule could be the following:

```
matching_rule(r1, ['dc-vdu','epa-attributes','host-epa','memory-mb'],
                  ['node_cluster','memory_mb'], >=)
```

The approach has a number of advantages. It can accommodate any binary relational operator, or even arbitrary Boolean binary functions. Different *matching engines* can be used. It allows reusability in the sense that it provides a clear separation between the NECOS information model and the providers' infrastructure representation, thus allowing Providers to simply adapt the mechanism by changing a relatively small number of rules. Finally, given an unsuccessful outcome, *i.e.*, failure to match a request, it can provide explanations based on the rules that failed. For example, the mechanism can reply that there is no host with the specific amount of memory requested, by reporting back violated matching rules to the broker. Such information can be used by NECOS or the tenant to *reformulate* resource requests in cases when all agents replied negatively and can be an easy-to-provide feature of the matching engine.

The matching mechanism is also responsible for the translation of the relational expressions that define tenant's constraints on resources as stated in the previous sections. These expressions override any operators indicated in the rule, since they express explicit tenant requirements. For instance, if for some reason the request includes the field `storage-gb: {in_range: [2, 4]}` then the corresponding rule will succeed only when a resource with storage in that range is found.

With minor extensions, the rule-based approach can also be employed to select the best offer on the providers' side. Given that it can be easily the case that multiple resources on a single provider can match a request, information regarding rule matching can provide the necessary input in order for the Provider's Agent to *select* the "best" resource offering: the latter can be formulated as an optimization problem seeking the best match of a request to an available resource. In this selection process, current infrastructure resource utilization, marketing strategies, and cost (as described in the corresponding section) can be incorporated in the objective function resulting in an "efficient" mechanism for the Provider.

For instance, consider that a request sent to a DC-Slice agent has the following fields (we omit sections of the request message, for reasons of clarity and presentational economy):

```
dc-slice-part:
    vdus:
     dc-vdu:
       name: core_vm
       epa-attributes:
         ...
         host-epa:
           cpu-number: 2
           memory-mb: 4096
```

```
          storage-gb: 2
      host-count-in-dc: {equal: 5}
```

Figure 23. Resource specification.

For example, we assume that the DC provider has clusters of similarly equipped machines, for instance, a cluster named ZOTAC and a cluster named DSS/MOBILE, and that the *Slice Agent* wishes to allocate the request on a single cluster, thus the problem reduces to finding the cluster with available hosts that matches the request specifications and provide a response for each cluster. Figure 24 indicates the information that a successful response should contain.

```
dc-slice-part:
   name: core_vm,
   allocated_to: ZOTAC
   nodes:5
   outcome:
     - succeeded:
        path: [dc-vdu, epa-attributes,host-epa,memory-mb],
        offered: 4096
        desired: 4096
     - succeeded:
        path: [dc-vdu,epa-attributes,host-epa,cpu-number]
        offered: 2
        desired:2
     - succeeded:
        path: [dc-vdu,epa-attributes,host-epa,storage-gb]
        offered: 160
        desired: 2
```

Figure 24. Successful matching response.

The reader should note that the mechanism reports for each rule, the requested (desired) and offered values of the resource characteristic. The difference between these two values can be used to select the "closest" to the requested resource. Different approaches can be used by the DC provider: For instance, the provider can evaluate alternatives based on a weighted sum of differences between desired and offered values, and select that minimizes this sum, or simply "promote" use of hosts in a cluster that is underused, thus implement any policy that is in accordance with its own business model.

A failure message is presented in Figure 25. The field that causes the failure is clearly indicated along with the values that caused the constraint violation. In the case that all alternatives for a provider are "failed", this explanation can be communicated back to the Broker in order to inform the builder *why* it was not able to successfully fulfil a request.

```
dc-slice-part:
   name: core_vm,
   allocated_to: DSS/MOBILE
```

```
nodes:5
outcome:
  - succeeded:
     path: [dc-vdu,epa-attributes,host-epa,memory-mb]
     offered: 4096
     desired:4096
  - failed:
     path: [dc-vdu,epa-attributes,host-epa,cpu-number]
     offered: 1
     desired: 2
  - succeeded:
     path: [dc-vdu,epa-attributes,host-epa,storage-gb]
     offered: 60
     desired: 2
```

Figure 25. Failed matching response

Although the responses follow the YAML format, the specific mechanism implementation is free to choose any internal representation. These responses will be used to formulate the answer of the *Slice Agent* to the *Broker*. Implementation details of a prototype matching engine will be presented in the deliverable D5.2.

4.3 Cost Model

We hereby present a mathematical formulation on how to extract the cost of a slice based on its requirements and a single possible answer set of slice parts, as well as a use case example of how to instantiate the proposed model. It is worth to mention that this is a first step towards a complete cost model to be used by the Marketplace, simple enough to be implemented into the prototypes and generic enough to be further extended in any suitable way.

4.3.1 Model Formulation

Firstly, we define a tenant's request for a slice s with a set of requirements as:

$$\mathcal{R}_s = \langle t_s^r, t_s^b, t_s^e, \mathcal{SP}_s, \mathcal{K}_s \rangle$$

where t_s^r, t_s^b and t_s^e represent respectively the request, beginning and ending times for the slice; $\mathcal{SP}_s$ represents the necessary slice parts for composing the slice; and $\mathcal{K}_s$ is a set of constraints. Explaining the latter first, we assume the $\mathcal{K}_s$ as the set of cost constraints per kind of resources. For instance, the maximum value a tenant is willing to pay for a host, link or network element. For this cost model, we do not consider any other type of constraints, such as geographic or performance ones. However, further extensions of the model with other types of constraints are highly desirable. A detailed instance of such a set of constraints $\mathcal{K}_s$ will be shown in a numeric example at the end of this subsection.

In order to define the slice parts, let us first consider the three sets of possible hosts, links and network elements, respectively, $H = \{h_1, h_2, h_3, ...\}$, $L = \{l_1, l_2, l_3, ...\}$ and $N = \{n_1, n_2, n_3, ...\}$. It is important to highlight that these sets represent the possible classes of elements which can be found in the infrastructure providers. So, h_k is assumed to be a host that has at least the capacity denoted by the class k, in terms of CPU, memory and disk storage. Finally, we consider that, when a tenant asks for a host, providers can offer any available host compatible but not inferior to class k. An inferior class of resource might have a lower price, however, might not be as efficient as the target class. It is also reasonable to think that a provider will offer the immediate higher class available for the sake of economy. The same considerations apply for links and network elements. So, we can define the slice parts of a specific slice s as multi-sets given by:

$$\mathcal{SP}_s = \begin{cases} \{(h, m(h)) : h \in H_s\}, & \text{where} \quad H_s \subseteq H \\ \{(l, m(l)) : l \in L_s\}, & \text{where} \quad L_s \subseteq L \\ \{(n, m(n)) : n \in N_s\}, & \text{where} \quad N_s \subseteq N \end{cases}$$

and $m(\cdot)$ is the multiplicity of a certain element inside the multi-set. Namely, the number of occurrences of the host element h_k in the multi-set is the number $m(h_k)$, and the same applies for links and network elements. A hypothetical possible representation of requirements for slice parts of slice s can be exemplified by:

$$\mathcal{SP}_s = \{\{(h_1 : 2), (h_2 : 2), (h_3 : 1)\}, \{(l_1 : 1), (l_2 : 1)\}, \{(n_1 : 1)\}\},$$

two hosts class 1, two hosts class 2 and one host class 3; one link 1 and one class 2; and one network element class 1.

In terms of the basic resources, we represent them by the main properties comprising each kind of resource. For this work, we consider that the main properties which define a host are CPU, memory and storage capacity. For the case of links, they are represented by capacity, delay and jitter properties. Lastly, network elements are considered by their capacity, ports and queue properties. The three types of resources are represented by the following tuples:

$$h_k = \langle cpu_k, mem_k, str_k \rangle, \forall k \quad | \quad h_k \in H$$

$$l_i = \langle capacity_i, delay_i, jitter_i \rangle, \forall i \quad | \quad l_i \in L$$

$$n_j = \langle capacity_j, ports_j, queue_j \rangle, \forall j \quad | \quad n_j \in N$$

Taken these denotations into consideration, we define the pricing functions that establish the prices of resources as a function of the resource class per time. For the host case, we represent the host k price as a function of its class and time as $f_p(h_k, t)$, from provider p. In a very similar way, the price functions for both link and network elements are given by $f_p(l_i, t)$ and $f_p(n_j, t)$ respectively. It is likely that infrastructure providers have the expertise to categorize their resources in order to apply distinct values for classes of resources.

Let $P = \{p_1, p_2, p_3, \dots\}$ be a set of providers, which one with potential resources to offer. To represent a single possible answer for a slice parts' request we define the multi-set $\mathcal{SP}_a$:

$$\mathcal{SP}_a = \begin{cases} \{(p, h, m_p(h)) : p \in P_{H_s} \text{ and } h \in (H_s \cap P_{H_s})\}, & \text{where} \quad P_{H_s} \subseteq P \\ \{(p, l, m_p(l)) : p \in P_{L_s} \text{ and } l \in (L_s \cap P_{L_s})\}, & \text{where} \quad P_{L_s} \subseteq P \\ \{(p, n, m_p(n)) : p \in P_{N_s} \text{ and } n \in (N_s \cap P_{N_s})\}, & \text{where} \quad P_{N_s} \subseteq P \end{cases}$$

where:

$$P_{H_s} = \{p : \text{p is in a unique set of P that can provide} \quad (H_s, m)\},$$

$$P_{L_s} = \{p : \quad \text{p is in a unique set of P that can provide} \quad (L_s, m)\},$$

$$P_{N_s} = \{p : \quad \text{p is in a unique set of P that can provide} \quad (N_s, m)\}.$$

More specifically, P_{H_s} denotes any subset of P that can provide exclusively the slice s hosts' request (H_s, m). It is important to highlight that, even though one provider has a specific host in H_s, it may not be part of the answer if: (1) the provider itself is the only one offering that host but does not have the required amount to offer; or (2) the answer has already covered the total amount for that host. So, that is why an answer should not only take into account the classes of the hosts, but also the amount required. Again, the same applies to the cases of links and network elements.

The total cost of a slice S, which takes into account the slice requirements $\mathcal{R}_s$ and a single possible answer $\mathcal{SP}_a$ is given by the following equation:

$$\begin{aligned} C_s(\mathcal{R}_s, \mathcal{SP}_a) = & \sum_{p \in P_{H_s}} \sum_{h \in (H_s \cap P_{H_s})} m_p(h) \cdot \int_{t_s^b}^{t_s^e} f_p(h,t)dt \ + \\ & + \sum_{p \in P_{L_s}} \sum_{l \in (L_s \cap P_{L_s})} m_p(l) \cdot \int_{t_s^b}^{t_s^e} f_p(l,t)dt \ + \\ & + \sum_{p \in P_{N_s}} \sum_{n \in (N_s \cap P_{N_s})} m_p(n) \cdot \int_{t_s^b}^{t_s^e} f_p(n,t)dt \end{aligned}$$

4.3.2 Cost Model Assessment

It is important to note that the number of possible solutions can vary a lot, depending on the number of providers and the set of requirements. The idea of this assessment example is to show how the cost model can be instantiated in a very simple scenario. Further evaluations using the cost model will be carried out, taking into account how each proposed parameter impacts the cost of a slice. Last but not least, we assume NECOS working *Mode 0*, where only dedicated resources are offered/traded by the Marketplace.

In this numerical example, for the sake of simplification, we consider the providers apply a very simple pricing function in order to value their resources. The price functions express a cost per unit of time over a specific class of resource, *e.g.*, x euros per hour for a specific host or y euros per hour for a specific network element. This is an oversimplification and a more elaborate analysis, involving more complex pricing functions, is a target for future work.

Given the above assumption, we consider two sets of providers represented by Table 1 and Table 2, illustrating hosts and connectivity providers respectively. Table 1 shows the specification of available hosts in two providers, as well as, the amount and price per each available host. We consider these two providers do not offer connectivity resources. On the other hand, Table 2 presents the only provider in this hypothetical example that offers connectivity resources and how they are priced.

It is worth mentioning that all the specifications, values, and prices used in this numerical example are hypothetical, just to demonstrate different situations in the assessment of the cost model. There is no real meaning or any logic behind the numeric values. Again, it is also a target for future investigation on how cloud slice providers (should) value their resources.

Host	Number	Price
Provider 1		
2 cores, 16 GB RAM, 128 SSD	2	€ 0.10 per hour
4 cores, 16 GB RAM, 256 SSD	3	€ 0.15 per hour
Provider 2		
2 cores, 16 GB RAM, 128 SSD	1	€ 0.05 per hour
4 cores, 32 GB RAM, 256 SSD	1	€ 0.15 per hour
8 cores, 64 GB RAM, 512 SSD	1	€ 0.25 per hour

Table 1. Computing providers' availability.

Resource	Number	Price
Provider 3		
Link: 1 Gbps, average delay 0.5 ms, jitter 0.10	2	€ 0.5 per hour
Switch: Gigabit Ethernet, 12 ports, WRR queue	1	€ 0.5 per hour

Table 2. Connectivity provider availability.

Now let us define a set of requirements for a target slice s, namely $\mathcal{R}_s$, comprising of two hosts, two links and one network element. Firstly, we consider a pretended slice request for the time window from Christmas eve until the end of the first day of 2020; *i.e.*, a 9 days, 216 hours or 12960 seconds time window. In the mathematical formulation, both t_s^b and t_s^e can be specified in terms of the request time t_s^r. From the defined slice parts comprising the request, we assume two hosts and two links with identical properties, and consider:

$$\mathcal{SP}_s = \{\{(h_1 : 2)\}, \{(l_1 : 2)\}, \{(n_1 : 1)\}\}$$

where the host, link, and network element specifications are as follows.

$$h_1 = \langle 2 \text{ CPU cores}, 8 \text{ GB RAM}, 64 \text{ GB disk} \rangle$$
$$l_1 = \langle 1 \text{ Gbps}, 0.5 \text{ delay threshold}, 0.1 \text{ jitter threshold} \rangle$$
$$n_1 = \langle 1 \text{ Gigabit Ethernet Switch}, 12 \text{ ports}, \text{any queue} \rangle$$

Finally, to complete the requirements for slice s, let us define the constraints $\mathcal{K}_s$ as the upper limit cost for usage of resources: € 3.00 per physical machine per day, € 15.00 per link per day, and € 15.00 per network element per day. It is worth mentioning that these kinds of cost constraints are already being used in the YAML specification files of the NECOS Marketplace, as well as, the specification of start and end times of a slice.

Given the definition of the above scenario, it is clear that the tenant wishes to pay at most € 459 for the usage of the whole slice. Moreover, given the defined providers' availability and tenant's constraints per each kind of resource, the only two sets of possible answers are:

$$\mathcal{SP}_{a_1} = \{\overbrace{\{(p_1 : h_1 : 2)\}}^{P_{H_s}}, \overbrace{\{(p_3 : l_1 : 2)\}}^{P_{L_s}}, \overbrace{\{(p3 : n_1 : 1)\}}^{P_{N_s}}\}$$

and

$$\mathcal{SP}_{a_2} = \{\overbrace{\{(p_1 : h_1 : 1), (p_2 : h_1 : 1)\}}^{P_{H_s}}, \overbrace{\{(p_3 : l_1 : 2)\}}^{P_{L_s}}, \overbrace{\{(p3 : n_1 : 1)\}}^{P_{N_s}}\},$$

where the former has an offer to deploy two hosts at Provider 1 (eventually in different domains), two links and one switch at Provider 3; and the latter differs by offering one host at Provider 1 and one host at Provider 2. Taking all definitions and constraints into account, the total costs of both answers $C_s(\mathcal{R}_s, \mathcal{SP}_{a_1})$ and $C_s(\mathcal{R}_s, \mathcal{SP}_{a_2})$ are respectively €367.20 and €356.40.

With this simple assessment example, it is possible to envision how the cost model can be used to evaluate the best offer for a slice, given a specific set of requirements. However, a more complex analysis is needed to assess the cost model, in terms of the three main perspectives it captures: (i) how the providers can better express the value of their resources in time, with different pricing functions; (ii) how the cost model is influenced by a varying number of providers; and (iii) how the set of constraints affects the cost of a slice. A sensitivity analysis of which parameters most affect the cost model is also desired. Finally, the cost model will evolve into an optimization problem in order to compute the best solution among all feasible ones.

4.4 Performance Evaluation

This section provides early results from the NECOS Marketplace prototype implementation. More specifically, we present experimental results related to the efficiency and the performance of the Marketplace.

4.4.1 Marketplace Efficiency

The NECOS platform implements the *Slice-as-a-Service* model, enabling the dynamic creation of end-to-end (E2E) slices from a set of constituent slice parts contributed from multiple domains. A challenging issue is to define the facility that implements dynamic slice resource discovery, aligned to the requirements of the slice owner or tenant, over different infrastructure providers. The NECOS Marketplace facility implements relevant federated interactions for the resource discovery based on the information model and the workflows presented in Sections 3.1 and 4.1, respectively. Hereby, we present early quantitative and qualitative experimental results validating the main operation of the NECOS Marketplace towards handling slice requests over different infrastructure providers.

In detail, our experimental setting consists of six different testbeds participating in the FED4FIRE federation [WAUTERS2014]: (i) w-iLab2, Virtual Wall 1 (VWall1), Virtual Wall 2 (VWall2) and Grid5000 test-beds, which are located in Europe; and (ii) CloudLab test-beds in Utah (ClabUtah) and Wisconsin (CLabWisconsin), *i.e.,* located in the USA. For the purposes of the experiment, each testbed is considered as a different infrastructure provider, with its own *Slice Agent* responding to slice part resource requests. We collected resource infrastructure data from the above testbeds, *i.e.,* regarding node availability and resource characteristics, such as memory, storage and bandwidth capacities, number of CPU cores. That data form the resource infrastructure information base of each agent, which is the information used by each *Slice Agent* to match particular resource requests incorporating extended EPA characteristics, as they are specified in the NECOS information model detailed in Section 3.

To evaluate the NECOS Marketplace facility, we generated slice requests that consist of a varying number of slice parts, each composed of one or multiple service function resource specifications, referred to as *dc-vdu* (*Data Center Virtual Deployment Unit*). Each *dc-vdu* specification is characterized by particular resource request demands, *i.e.,* number of physical nodes, amount of RAM, disk storage, bandwidth capabilities. Assuming that each testbed serves as a single datacenter (DC) infrastructure provider entails that it can accommodate only one (DC) slice part. The testbed resources are organized in clusters of similar in characteristics computing nodes, so we make the assumption that each *dc-vdu* can be accommodated in a single node cluster. Thus, the question that each *Slice Agent* answers is whether it can accommodate or not the request received by the *Broker* for a single slice part, allocating each *dc-vdu* of the part to possibly different node clusters in its infrastructure.

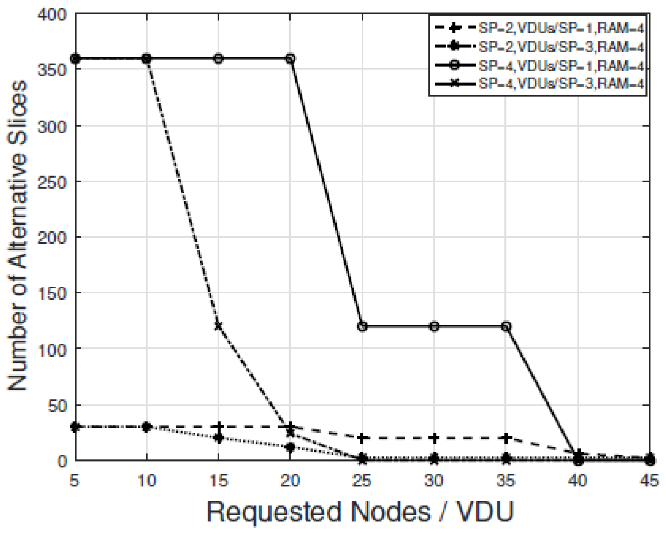


Figure 26. Number of alternative slices with one resource requirement.

We organize our experimental evaluation in three series of trials to investigate the solution space of the Marketplace based on various requested slices, *i.e.,* the first two with quantitative and the third with qualitative results. In the first series, we adjust the slice request by increasing the number of physical nodes within a *dc-vdu*, the latter annotated with a single resource specification, *i.e.,* the memory size (RAM). In the second series of experiments, we generate slice requests of similar demands in node availability, but with three resource specifications for each *dc-vdu* (*i.e.,* RAM, disk storage and bandwidth capabilities). Finally, the last experiment involves a cloud core / edge slice request example and demonstrates its geographical distribution.

In the first and second series of results, we validate the overall functionality of the marketplace and obtain an idea on the size of the solution space. Essentially, we vary the demands in slice requests as described above and compare the number of alternative slice solutions for scenarios with a different number of slice parts (*SP=[2,4]*), and *dc-vdus* per slice part (*VDUs/SP=[1,3]*). Results of the first series are depicted in Figure 26, where we observe that the number of alternative solutions decreases as the requested nodes per *dc-vdu* increases. Such behaviour is expected, as the number of providers that can accommodate the slice part decreases. Note that the number of parts plays a significant role in the solutions, since it allows for more combinations of providers (*i.e.,* alternatives) to be generated. Similar results hold for the second series of experiments, as depicted in Figure 27. An interesting point to note here is that in slices with a high number of parts and an increased set of resource requirements, the decrease in the number of solutions is sharp. This is attributed to the fact that very few infrastructure providers can accommodate resource-demanding slice parts.

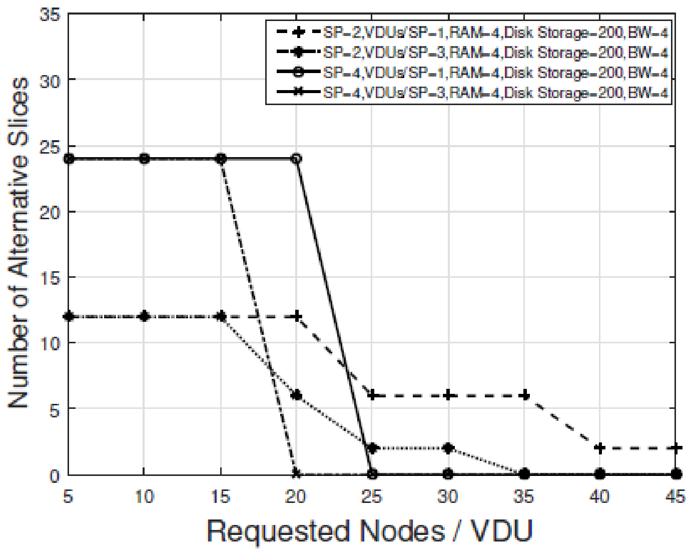


Figure 27. Number of alternative slices with three resource requirements.

It is important to emphasize that, from the first two series of results shown in Figure 26 and 27, we can notice the curves in both graphs behave the same, where the number of alternative solutions decreases as the demand for requested nodes increases. Even with the increase in slice parts, which provides more alternative combinations, the number of possible slices drops significantly from 20 requested nodes per *dc-vdu* onwards. Finally, as also expected, the more requirements for resources (three instead of one), the less is the number of alternative slices.

The final experiment aims to demonstrate the allocation of a slice to a set of core / edge cloud providers. The slice consists of four slice parts, each hosting two *dc-vdus*. The slice parts have diverse resource demands, reflecting their role in the slice, *i.e.*, higher demands for the core cloud slice resources and lower demands for the edge cloud resources. The following example presents the constraints for a core cloud slice:

```
dc_part(dc-slice-1,location:'undefined',
 ['dc-vdu'(vdu11,[
   'available-nodes'>=2,'memory-gb'>64,
   'min-storage-gb'> 250,'nics-bw'> 2]),
  'dc-vdu'(vdu12,[
   'available-nodes'>=1,'memory-gb'>=128,
   'min-storage-gb'>=500,'nics-bw'>=5])
 ])
```

Figure 28. Core cloud slice constraints.

whereas the example indicated below shows the constraints for a slice part corresponding to an edge cloud, where resources are less demanding:

```
dc_part(dc-slice-2,location:'undefined',
 ['dc-vdu'(vdu21,[
   'available-nodes'>=1,'memory-gb'>=8,
   'min-storage-gb'>=80,'nics-bw'>=1]),
  'dc-vdu'(vdu22,[
   'available-nodes'>=1,'memory-gb'>=8,
    'min-storage-gb'>=80,'nics-bw'>=1])
 ]),
```

Figure 29. Edge cloud slice constraints.

Slice part 1	Slice part 2	Slice part 3	Slice part 4
Grid5000	CLabUtah	CLabWisconsin	VWall2
Grid5000	CLabUtah	VWall1	VWall2
Grid5000	CLabUtah	w-iLab2	VWall2
Grid5000	CLabWisconsin	CLabUtah	VWall2
Grid5000	CLabWisconsin	VWall1	VWall2
Grid5000	CLabWisconsin	w-iLab2	VWall2
Grid5000	VWall1	CLabUtah	VWall2
Grid5000	VWall1	CLabWisconsin	VWall2
Grid5000	VWall1	w-iLab2	VWall2

Table 3. Slice part allocation to different infrastructure providers.

Similar constraints exist in the other two parts. In the current experimental setup, the marketplace generated nine alternative allocations for the slice, as depicted in Table 3. Note that since the request did not define any geographic constraints on the slice parts, the marketplace generated all possible solutions, distributing slices between Europe (Grid5000, w-iLab2, VWall1 & VWall2) and USA (CLabUtah & CLabWisconsin). This demonstrates that such a loosely coupled resource discovery model can manage a diverse set of geographically distributed providers.

4.4.2 Marketplace Performance

We further present performance measurements associated with the Marketplace operations, *i.e.*, the discovery and the selection of resources for slice instantiation. Our prototype implementation of the NECOS architecture encompasses five slice deployment steps, namely *Resource Discovery and Selection* (which corresponds to the Marketplace operations), *Physical Resource Allocation*, *Slice Stitching*, *Service Deployment*, and *Monitoring Activation*. For the purpose of the experiment that follows, we consider a distributed content service that geographically spans over Europe and the USA. We assume that a *Tenant* requests a slice consisting of the following service functions: (i) a cluster of Web servers, (ii) a service load balancer (SLB), and (iii) benchmarking tools (BT). We further consider that the *Tenant* specifies each service function to be allocated in a particular geographic location, and thus, the service functions span two different DC slice segments, as depicted in Figure 30. In particular, the left DC slice segment contains resources from the FED4FIRE testbeds federation, whereas the right DC slice segment consists of physical resources located at our own UOM testbed. To emulate the WAN slice segment that stitches the DC slice segments, we deploy an additional physical machine at each side, acting as an edge router.

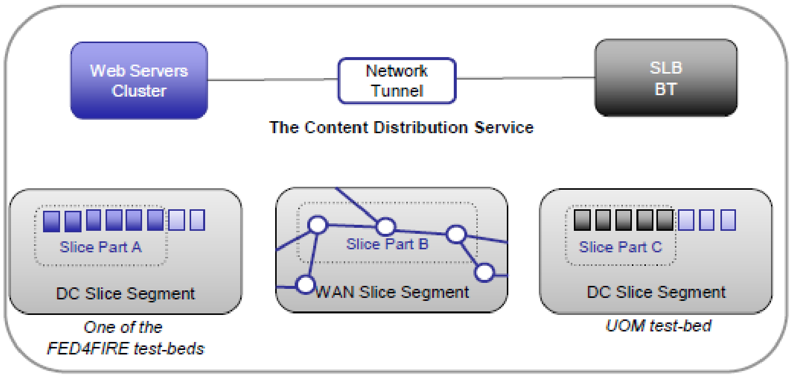


Figure 30. Content distribution service slice.

To proceed with the slice deployment, we defined the service and slice requirements as a generic YAML slice-information input that includes the three slice segments. In more detail, the DC slice segment at the UOM testbed consists of six physical nodes hosting: (i) a service load balancer which distributes (*i.e.,* in a round-robin fashion) the Web traffic of a number of clients to the Web servers located at the left-side DC slice segment; and (ii) the benchmarking tools emulating the clients' behaviour. The DC slice segment with the Web servers' cluster is physically located in the USA (*i.e.,* the CloudLab Utah testbed) and is accessed through the FED4FIRE facility. The number of physical machines in this segment is expressed by the parameter *Nodes* of our experiment, which is in the range of [5 … 30]. We choose two classes of *Nodes'* hardware type, *i.e.,* the *pc3000* class with *3.0 GHz* processor, *2 GB* DDR2 RAM and *300 GB* storage, and the *d430* class with two *2.4 GHz 8-core* processors, *64 GB* DDR4 RAM and *2.2 TB* storage[2]. The first class could serve as edge cloud and the latter as core cloud.

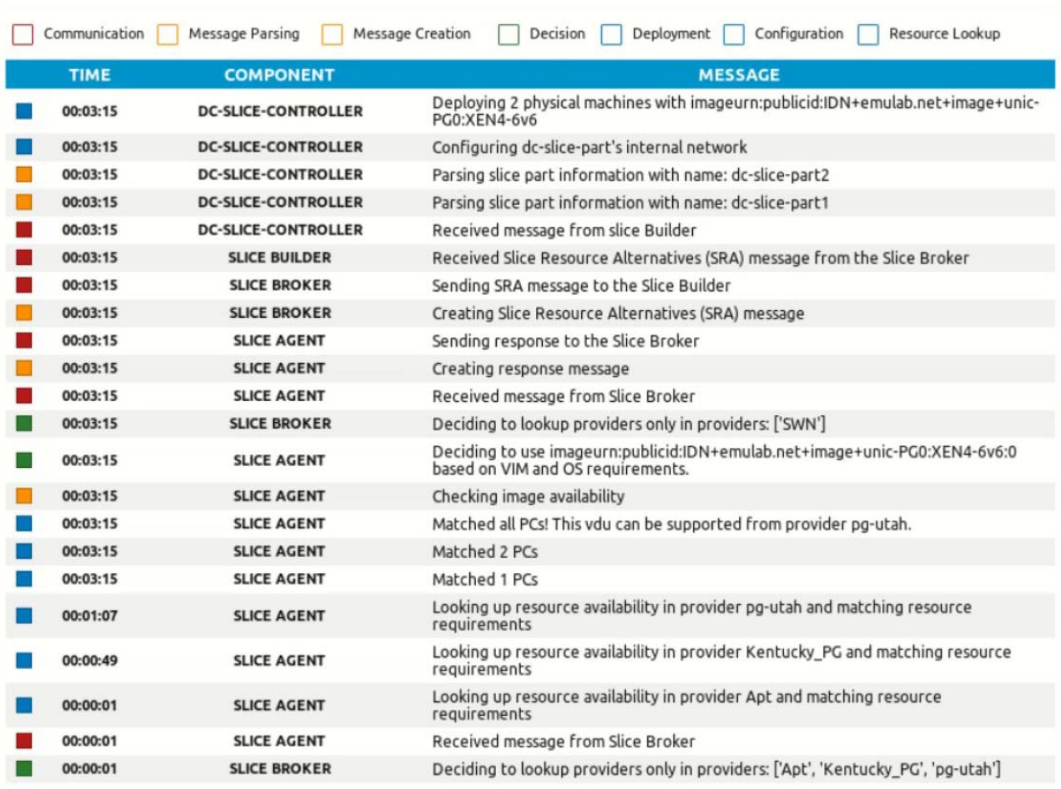
In this particular DC slice segment, we consider the deployment of a Web server per physical node, we define the virtualization technology, and further specify the service resource flavour (*i.e.,* CPU, RAM utilization and storage usage) for each Web server. Furthermore, we designate the traffic policy (*i.e.,* equally, randomly) for the service. The third slice segment (WAN) is responsible for configuring the inter/intra-domain connectivity of the DC slice segments. We boot an extra physical machine in each DC slice side, which acts as an edge router, and we set up a GRE tunnel between them to configure their connectivity. To secure intra-connectivity, we configure each physical node's routing table to allow the communication with the edge router of the other side. For inter-domain connectivity, we assume a star topology where each physical node is connected to the edge router (central node) and through the edge router to the remote DC slice segment (remote physical nodes).

For the *Monitoring Activation* step of slice deployment, we perform the configuration of the CollectD open-source monitoring tool. For the allocated physical resources, we enable the following KPIs: (i) CPU and RAM usage, and (ii) incoming/outgoing traffic. The tool collects the KPI metrics every *20 sec* (time interval).

To track the outcome of all the slice deployment processes, we developed the dashboard debug window, shown in Figure 31, where we recite the events bottom-up. We group up the processes of the involved NECOS components in different categories, *i.e.,* Communication, Message Parsing, Decision, Deployment, Configure, Resource Lookup. Since an experiment's outcome is lengthy, in Figure 31 we only display a portion of it. Indicatively, we can distinguish a resource discovery operation looking for a provider geographically located in the USA (*i.e.,* one of the supported Apt, Kentucky or CloudLab Utah testbeds) and, during the matching process, the system decides to discard the first two options.

[2] A detailed description of hardware specifications can be found at: https://wiki.emulab.net/wiki/UtahHardware

This is because we predefined that one of the DC slice segments should be deployed in the USA and the requested resources can only be found in the CloudLab Utah testbed.



□ Communication □ Message Parsing □ Message Creation □ Decision □ Deployment □ Configuration □ Resource Lookup

	TIME	COMPONENT	MESSAGE
🟦	00:03:15	DC-SLICE-CONTROLLER	Deploying 2 physical machines with imageurn:publicid:IDN+emulab.net+image+unic-PG0:XEN4-6v6
🟦	00:03:15	DC-SLICE-CONTROLLER	Configuring dc-slice-part's internal network
🟧	00:03:15	DC-SLICE-CONTROLLER	Parsing slice part information with name: dc-slice-part2
🟧	00:03:15	DC-SLICE-CONTROLLER	Parsing slice part information with name: dc-slice-part1
🟥	00:03:15	DC-SLICE-CONTROLLER	Received message from slice Builder
🟥	00:03:15	SLICE BUILDER	Received Slice Resource Alternatives (SRA) message from the Slice Broker
🟥	00:03:15	SLICE BROKER	Sending SRA message to the Slice Builder
🟧	00:03:15	SLICE BROKER	Creating Slice Resource Alternatives (SRA) message
🟥	00:03:15	SLICE AGENT	Sending response to the Slice Broker
🟧	00:03:15	SLICE AGENT	Creating response message
🟥	00:03:15	SLICE AGENT	Received message from Slice Broker
🟩	00:03:15	SLICE BROKER	Deciding to lookup providers only in providers: ['SWN']
🟩	00:03:15	SLICE AGENT	Deciding to use imageurn:publicid:IDN+emulab.net+image+unic-PG0:XEN4-6v6:0 based on VIM and OS requirements.
🟧	00:03:15	SLICE AGENT	Checking image availability
🟦	00:03:15	SLICE AGENT	Matched all PCs! This vdu can be supported from provider pg-utah.
🟦	00:03:15	SLICE AGENT	Matched 2 PCs
🟦	00:03:15	SLICE AGENT	Matched 1 PCs
🟦	00:01:07	SLICE AGENT	Looking up resource availability in provider pg-utah and matching resource requirements
🟦	00:00:49	SLICE AGENT	Looking up resource availability in provider Kentucky_PG and matching resource requirements
🟦	00:00:01	SLICE AGENT	Looking up resource availability in provider Apt and matching resource requirements
🟥	00:00:01	SLICE AGENT	Received message from Slice Broker
🟩	00:00:01	SLICE BROKER	Deciding to lookup providers only in providers: ['Apt', 'Kentucky_PG', 'pg-utah']

Figure 31. NECOS debug window.

Our evaluation results show the slice deployment time when either core or edge cloud nodes are employed at the DC slice segment accommodating the Web servers' cluster. Figure 32 and Figure 33 provide both a general view of the total time incurred for slice instantiation, as well as the time spent for each individual step, *i.e., Resource Discovery/Selection*, *Physical Resource Allocation*, *Slice Stitching*, *Service Deployment*, and *Monitoring Activation*. This time is expressed as a function of the number of the physical *Nodes* deployed at the CloudLab Utah testbed. We report the results across five runs.

Figure 32 and Figure 33 indicate that the less time-consuming steps are *Resource Discovery/Selection*, *Slice Stitching* and *Monitoring Activation*. *Resource Discovery/Selection*, in particular, is almost fixed at around *30 secs* in the case of core cloud nodes, and at around *35-40 secs* in edge clouds. This corroborates the nice scaling properties of the Marketplace operations with resources allocated from multiple providers. The *Slice Stitching* step ranges from almost *20-120 secs* and *25-130 secs*, for core and edge cloud nodes, respectively. This step along with the monitoring increase linearly with the number of nodes. The *Monitoring Activation* starts at *16 sec* and reaches almost *100 secs* (Figure 32), whilst it ranges from *21* to *133 secs* in Figure 32 and Figure 33. These results show slightly lower delays when edge cloud nodes are allocated for the DC slice segment.

On the other hand, the *Physical Resource Allocation* and the *Service Deployment* steps are the most time-consuming. Resource allocation involves the servers' boot-up time, which entails the prolongation of the total deployment time. However, in the case of the core cloud nodes, the resource allocation requires as much as *74%* of deployment time when *Nodes=5*, which significantly decreases to *30%* when *Nodes=30*. The corresponding percentages range from *69-37%* in case of edge cloud nodes. This decrease is due to the fact that the time remains almost stable with the increase of the nodes. In contrast to this observation, the delay incurred for *Service Deployment* increases with the number of nodes. As a result, service deployment attributes *14-40%* and *16-38%* (in respect to the number of nodes) of the total slice deployment time when core and edge cloud nodes are employed, respectively.

In summary, the Marketplace operations attribute to only a small fraction of the overall delay incurred for slice instantiation. Furthermore, there is no perceptible increase in the delay incurred during resource discovery and selection, as the number of slice nodes increases. In essence, these performance measurements provide an indication for the performance and the scalability of the NECOS Marketplace.

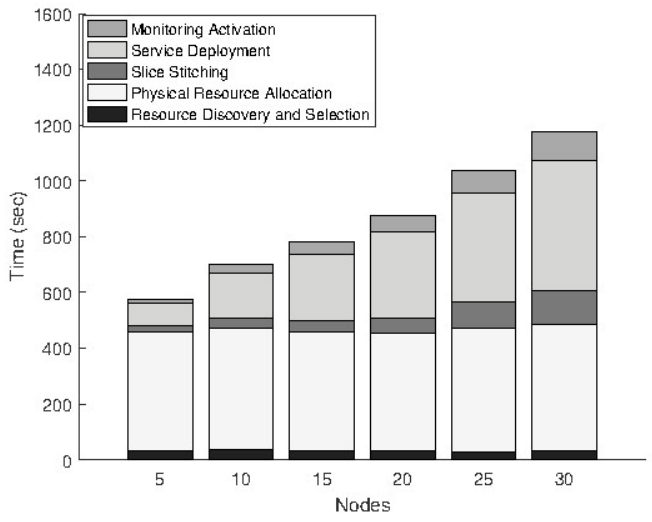


Figure 32. Slice deployment time for core cloud nodes.

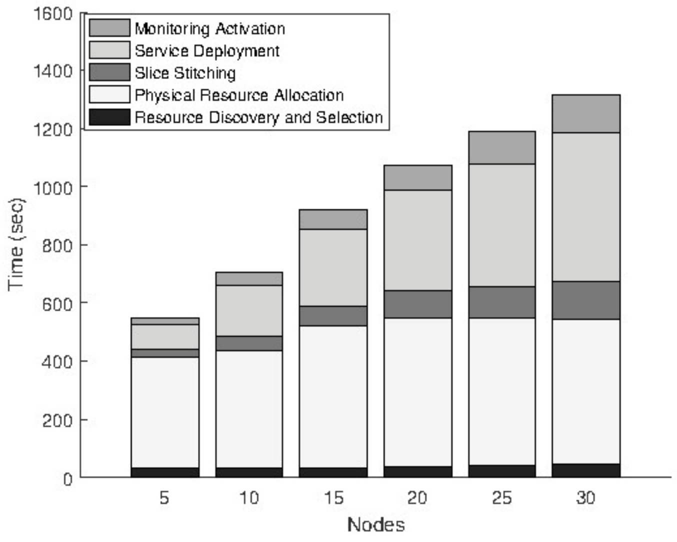


Figure 33. Slice deployment time for edge cloud nodes.

5 NECOS Cloud API Specifications

This section presents the cloud APIs specified by NECOS for slice request, creation, configuration, and run-time management. These cloud APIs have been classified into: (i) *client-to-cloud APIs*, which include API methods invoked by the *Tenant* for slice request, control and run-time management, and (ii) *cloud-to-cloud APIs*, which are associated with interactions between NECOS system components residing in different domains (*e.g.*, in the case of federation), such as the *Slice Resource Orchestrator*, the *Slice Builder*, *Slice Broker*, the *Slice Agents*, and the *Slice Controllers*, as shown in Figure 34. Section 5.1 elaborates on the *client-to-cloud API* specifications, whereas Section 5.2 describes the *cloud-to-cloud APIs*.

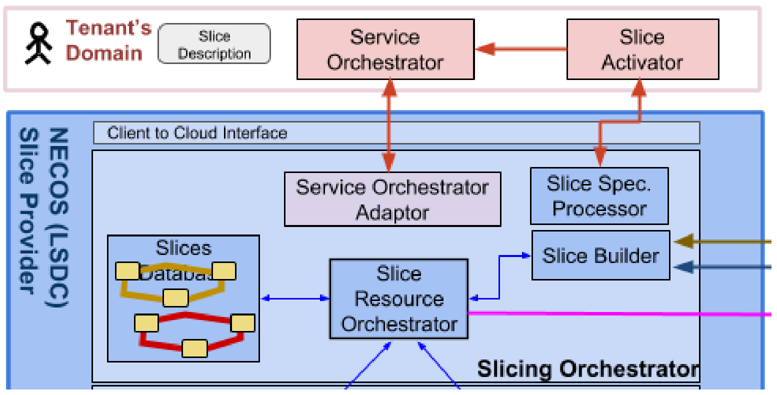


Figure 34. NECOS Client-to-Cloud Interface / API

5.1 Client-to-Cloud API

5.1.1 Slice (and Service) Management

NECOS provides different means to a *Tenant* to request the instantiation of a service and / or slice to a *NECOS (LSDC) Slice Provider*, as detailed in this section. The tenant has the option to initiate the creation of a slice with one of the following specification types, reflecting different levels of abstraction: (i) the Slice Specification, the lowest abstraction level focusing on resource aspects, (ii) the Slice Requirements, specifying the general slice requirements and leaving to the *Slice Builde*r with the assistance of the *Slice Specification Processor* to determine the slice details, and (iii) the Service Specification that describes only the (type of) service to be deployed and delegates to the NECOS system the identification of the characteristics of the best suitable slice to be allocated. As discussed in Section 3, the information model representation for the *Tenant* covers all these aspects and the above three types of slice requests complete different sets of attributes in the model. Subsequently, the *Slice Specification Processor* prepares the input of the *Slice Builder* and the latter translates the Slice Requirements or the Service Specification to an equivalent *Slice Specification*, in the form of the Partially Defined Template (PDT).

In particular, NECOS supports the following API methods for slice instantiation and management:

create_slice (Slice Specification): Slice ID

When this API call is invoked, the *NECOS Slicing Orchestrator* receives an explicit slice specification (also including slice requirements) as the input from a *Tenant*. This is the lowest level of abstraction that can be used by a Tenant to interact with the NECOS Slice Provider via the *Client-to-Cloud Interface*.

The API method **create_slice** shall be invoked by the *Tenant* (*e.g.*, via the *Slice Activator* in Figure 34) providing an explicit description of the slice topology to be instantiated (using the NECOS information model presented in Section 3, including *e.g.*, required resources such as number of cores, type of

VIMs/WIMs, geographical constraints of the infrastructure elements, specific monitoring subsystems to be deployed, etc.) along with potential expected slice requirements (*e.g.*, delay constraints, rules of compliance, etc.). As an example, when using this API call, the *Tenant* may explicitly define the number of (physical) hosts to be part of the slice, a particular type of VIM to be used to manage a slice segment, as well as slice access points to be used later on to anchor service instances to that slice (*e.g.*, hooks to the VIM or SSH connectivity for remote configuration tools, such as Ansible).

The submitted input parameters are processed by the *Slice Specification Processor*, which takes care of merging the requested slice topology description with the slice requirements, also verifying their full compliance. As soon as the above check is completed, the final slice specification is eventually produced and provided as an input to the *Slice Builder* component of the *NECOS Slicing Orchestrator*. The operation flow will then continue according to the steps required for building a (multi-provider) slice via the *Cloud-to-Cloud API*. The *Slice Builder* also takes into account the received slice requirements to generate a set of rules to be used by the *Slice Resource Orchestrator* to trigger slice lifecycle events – parameters of the slice are being adjusted at run-time according to monitoring information collected, while the slice is up and running using the *Cloud-to-Cloud API*.

At the beginning of the Slice creation process, a **Slice ID** associated to a new requested slice is generated by the NECOS Slicing Orchestrator, and will be utilised by the system to univocally identify that specific slice instance during both the slice creation process and when the slice will be up and running. The above **Slice ID** will eventually be returned to the *Slice Activator* in the *Tenant*'s Domain after the invocation of the **create_slice** call is completed. As the end-to-end slice creation process might take a non-negligible time to be completed, we designed the above call (and the invocation of the other overloaded **create_slice** API endpoints described later on in this section) to be asynchronous. After a call invocation, the module of the *Slice Activator* that processes the slice requests coming from the *Tenant* will, in fact, terminate its execution thread instead of waiting for the actual allocation of the requested slice. This will allow the *Slice Activator* to handle further requests that might be generated by the *Tenant* and to pass them on to the *NECOS Slicing Orchestrator* (trying to achieve a higher level of concurrency). Meanwhile, a separate module within the *Slice Activator* will listen for callbacks coming from the *NECOS Slicing Orchestrator*, which will provide a notification about the successful allocation of a previously requested slice. When an end-to-end (pending) slice is successfully created, the *Slicing Orchestrator* will in fact "call back" the *Slice Activator* that initially submitted the slice request, providing the **Slice ID** that was initially associated to that slice and a response status that specifies whether the slice allocation process was successful or not.

Once received the notification callback, the *Slice Activator* will propagate both the **Slice ID** and the result of the slice creation to the *Tenant*, who will have the option of either using that ID directly in their own domain space, or to perform an internal mapping (on locally generated IDs) if the ID format expected by *e.g.,* the Service Orchestrator is different. The (mapped) ID will then be utilised by the *Tenant* to deploy service instances on that particular slice via its own *Service Orchestrator*.

Alternatively, a *Tenant* may provide the description of a slice to be instantiated by the *NECOS Slice Provider* at a given future instant in time. In this case, the *Slice Specification* descriptor will include the additional internal fields *Start Time* and *End Time*. Although the operational flow will not change in respect to the previous **create_slice** API call, this time the *Slice Builder* will not actually activate the slice until the specified time instant is reached (*i.e.*, specified by the *Start Time* parameter) and the slice will stay active until the time instant specified by the *End Time* parameter will be reached. However, resource reservation might be requested on the potentially involved domains via the *Slice Marketplace Cloud-to-Cloud interface*. As in the case of the immediate (*i.e.*, not scheduled in the future) slice instantiation request described above, the *Slice Activator* will be notified via callback by the *Slicing Orchestrator* when the end-to-end slice instantiation process is (successfully) completed and the slice will actually be up and running.

create_slice (Slice Requirements): SliceID

In this case, the *Slice Resource Orchestrator* receives a set of Slice Requirements from a *Tenant* that wishes to request the allocation of a given set of resources, in the form of slice, for its services to be instantiated. However, the *Tenant* is aware of the service's requirements and uses them as an input for the novel *NECOS Slice-as-a-Service* feature. As such, the *Tenant* is not providing a full specification of the slice elements, as in the previous API call. For instance, a requirement of the slice might be associated with delay bounds between two arbitrary elements of the slice to be created, *e.g.*, for delay-sensitive services, the expected delay must be lower than a given critical threshold specified as part of the desired QoS. That delay information might be factored in during the creation of the slice in order to *e.g.*, map the slice connectivity related part on the proper network links interconnecting the computation and storage resources.

In this case, a *Tenant* invokes the **create_slice** method and provides as a parameter a description of the expected slice requirements. The requirements are in the form of contract, an equivalent to Service Level Agreement for a Slice (*i.e.*, a Slice Level Agreement), consisting of one or more slice objectives (expressed as specific constraints on some KPIs of interest for the slice) to be fulfilled throughout the whole lifecycle of the slice. This API method invocation will again result in a request being sent to the *Slice Specification Processor* of the *NECOS Slicing Orchestrator*, which will process the requirements and generate a Slice Specification for the *Slice Builder*. The workflow will then continue as in the description of the previous API call according to the same callback-based, asynchronous mechanisms described there.

create_slice (Service Specification): Slice ID

In this case, a *Tenant* uses the NECOS *Client-to-Cloud Interface* to provide a high-level specification of a service (including also additional service-related requirements / KPIs objectives) that will then be translated by the Slicing Orchestrator into the actual Slice Specification required to run that particular (type of) service. The service may be defined as a forwarding graph that contains service elements (*e.g.*, VNF instances, VM / Container image types) available from different infrastructure providers / domains. This Service Specification is being considered together with the provided service requirements, at the slice instantiation time.

In this case, the *Slice Activator* in Figure 34 receives the *Tenant*'s Service Specification, in the same way with a Slice Specification and invokes the API call using that specification as an input parameter. This has the effect of propagating the Service Specification to the *Slice Specification Processor* within the *NECOS Slicing Orchestrator*. A Slice Specification related to that service is generated from the *Slice Specification Processor* and eventually processed by the *Slice Builder* to start the slice creation process. From that moment, the workflow is the same, as described for the previous API calls. As an additional step, once the slice creation is completed (i.e., a successful callback is initiated by the Slicing Orchestrator for the associated **Slice ID**), the *Slice Activator* will trigger the actual service instantiation on the new slice via the *Service Orchestrator* in the *Tenant*'s Domain.

delete_slice (Slice ID)

This API call is exposed to a *Tenant* to allow the deactivation of a slice that is no longer required. The invocation of this call triggers the decommission of the slice (identified by the **Slice ID**) via the interaction with the *Slice Resource Orchestrator*, which takes care of releasing the allocated resources in the corresponding Resource Domains (via the *Cloud-to-Cloud API*). In case there is a service running on the slice, the **delete_slice** method triggers the termination of the particular service.

get_slice_parts (Slice ID): Slice Part ID [ ]

As soon as a Slice is successfully instantiated, its related **Slice ID** has been generated and returned to the requesting *Tenant*, and a successful callback has been received associated to that **Slice ID**, further operations might be carried out on the slice in order to, *e.g.*, adjust its configuration and / or perform life-cycle operations on the slice. The *Tenant* may retrieve additional information about his allocated Slice using this API call by providing the ID of the Slice of interest as an input parameter. This call returns a set of IDs associated with the different *Parts of the Slice* that can then be used to perform fine-grained configuration / operational tasks on each element of the allocated slice. This call provides a lower-level view of the slice abstraction to the *Tenant* (*i.e.*, to have access to the slice parts) to allow better fine-tuning of the provided slice.

get_slice_topology (Slice ID): Slice Topology

The **get_slice_topology** API endpoint is similar to the above described **get_slice_parts** call. However, in this case the *Tenant* will be able to collect the detailed topology of a previously allocated slice using a single API invocation. When this API call is invoked, the different Parts of the Slice identified by the **Slice ID** parameter will first be retrieved; for each of the Slice Parts, all the allocated slice elements will then recursively be collected and aggregated in order to build a single **Slice Topology** descriptor, which will eventually be returned back to the requesting *Tenant*. The latter will use that descriptor, *e.g.,* to perform service orchestration operations that require the mapping of different service elements on the topology of a given end-to-end slice.

deploy_service ([Service Specification, Service Name], SliceID): Service ID

This API call is invoked by a *Tenant* to request the deployment of a service on the slice (identified by the **SliceID**) that was previously created by the *NECOS Slice Provider*. The service can be identified by its name if the Service Specification was used for the slice creation, or by a new Service Specification. In the latter case, this process includes a step for validating the consistency of the Service Specification with the Slice Specification of the particular slice. When this call is invoked by the *Service Orchestrator* in the *Tenant*'s Domain, the Service Specification is passed to the *Service Orchestrator Adaptor* (in the *NECOS Slicing Orchestrator*), which, in turn, performs any required adaptation before requesting the embedding of the desired service elements on the slice. The call returns the ID of the deployed service.

start_service (Service ID)

This API call is used by a Tenant, who previously submitted a successful service deployment request (identified by **Service ID**) via the **deploy_service** API call, to actually activate the deployed service elements and mark the service instance as "running".

stop_service (Service ID)

This API call is used by a *Tenant* who previously submitted a service instantiation request (for a service identified by the **Service ID**) to stop the corresponding service elements, deployed on a slice.

get_service_info (Service ID): Service Status Information

This API call is invoked by a *Tenant* who previously submitted a service instantiation request to retrieve information about the runtime status of that particular service (identified by **Service ID**). This information can include, *e.g.*, the status of the service elements (such as VNFs, containers, links) in the case where the service monitoring process was delegated to NECOS (according to the level of abstraction in the interaction *Tenant / Provider*).

5.1.2 Slice Configuration

After a successful slice provisioning requested by a *Tenant* using the API calls described in Section 5.1.1, the client should have the ability to configure and operate his slice. For example, the client may request the scaling of an existing slice, *e.g.,* the addition of new hosts to increase the compute / storage resources, links, or network elements (*e.g.,* virtual switches, routers, etc.). In this respect, NECOS provides *Client-to-Cloud API* methods that are associated with the slice configuration and its lifecycle management.

In terms of slice configuration, the *Client-to-Cloud API* provides different methods to be used according to the level of abstraction (*i.e.,* on the slice request or slice mode) that a *Tenant* wishes to use when interacting with a *NECOS Slice Provider*, as detailed in the two following subsections.

5.1.2.1 Lowest abstraction level

NECOS provides additional *Client-to-Cloud API* methods to *Tenants* that wish to exercise various operations (*e.g.,* control, configuration, life-cycle management) to a slice that was previously allocated by a *NECOS Slice Provider*. In the following, we present a set of such API methods that allow the *Tenant* to interact with the different slice parts and to the different elements related to them using the lowest available level of abstraction. In this section, we refer to element as a generic instantiated entity, representing, *e.g.,* a *Host*, a *Path*, a *Switch*, a *Router* (this list may not be exhaustive).

get_slice_part_infrastructure_management_handle (Slice Part ID): Slice Part Management URL

This API method allows a *Tenant* to provide the ID of a slice part and to retrieve (when available) the URL that should be used to interact with the management interface of the VIM / WIM running on that slice part. This can be, for instance, the URL of the GUI that should be accessed to customise an Openstack instance that was deployed (on-demand) in the slice part identified by the **Slice Part ID**.

get_slice_part_elements (Slice Part ID): Slice Part Element ID []

A *Tenant* that requires performing low-level configuration and management operations on the elements composing a slice part, uses this API call to retrieve the related references to them. When invoking this call using a given **Slice Part ID**, a list of **Element IDs** for that slice part are returned.

get_element_handle (Element ID): Element Management URL

This API call provides the abstractions to retrieve a proper Management entry point associated to the element of a slice part identified by an **Element ID**. For example, if the element is a physical host, a URL may be returned to access that particular host, e.g., for VNC or SSH.

add_element (Slice Part ID, Element Specification): Element ID

A *Tenant* uses this API call to dynamically modify the topology of an already allocated slice. This call provides a *Tenant* with a relatively abstracted way to add a new element to the slice part identified by the **Slice Part ID**. The NECOS components take care of reconfiguring the topology of the end-to-end slice to fulfil the *Tenant*'s request. The Element Specification contains information on the element, such as an image for the physical host, router configuration, etc. The *NECOS Slicing Orchestrator* forwards the **add_element** requests to the relevant *Slice Controllers*. The latter instantiate elements matching the element specifications and include them in the corresponding slice parts. In case the particular slice part does not have enough resources, this may trigger either a negative response or a slice elasticity process.

delete_element (Element ID)

As described for the previous API call (**add_element**), a *Tenant* is able to delete one of the elements from a slice part by providing the corresponding **Element ID**. The NECOS system abstracts the required management and configuration operations to the *Tenant* and the topology of the end-to-end slice is automatically modified to fulfil the *Tenant*'s request.

5.1.2.2 Intermediate abstraction level

In a slightly different scenario of interaction between the *Tenant* and a *NECOS Provider*, the former might need to modify / adjust the topology of an already existing slice without being aware of the related implementation details. In this case, the *Tenant* should consider using the API calls described in this subsection. In this abstraction level, the *Tenant* manages the slice as a whole, in a seamless way with respect to the consisting slice parts.

get_slice (Slice ID): Slice Specification

The *Tenant* uses this API method to retrieve the description (*i.e.*, the representation of the slice using the NECOS information model in Section 3) of a slice identified by its **Slice ID**.

update_slice (Slice ID, [Slice Specification], [Service Specification])

A *Tenant* that requested the allocation of a slice and received the associated **Slice ID** after its successful instantiation, is able to modify the slice topology by providing an updated Specification for it (either via the Slice or Service Specification parameter). The API call is processed by the NECOS system that transparently adjusts the arrangement of the slice parts (and elements) according to the delta between the existing and the old slice specifications. This abstraction level does not allow direct manipulation of service elements (i.e., addition or removal), because this may trigger inconsistencies in the Service Specification, so the **update_slice** method with a refined Service Specification should be used instead.

add_resources (Slice ID, Resource Descriptor)

This API method allows a *Tenant* to dynamically modify the topology of his own slice (identified by the **Slice ID**) by adding new (virtual) resources via the specification of a Resource Descriptor to be attached to the already existing slice. This is slightly different from the **update_slice** method, as in this case the explicit specification of the resource elements to be attached to the slice are being provided, instead of a new global slice description. The Resource Descriptor is the subset of the Slice Specifications that corresponds to one or a set of resources.

delete_resources (Slice ID, Resource Name)

A *Tenant* uses this API to explicitly delete resources from the slice identified by the **Slice ID** and described by the **Resource Name**, which is an attribute of the Resource Descriptor specified through the previous call. The NECOS system processes the request and abstracts the operations related to the modification of the existing slice. The **Resource Name** should be unique within a particular slice.

5.1.3 Slice Monitoring

get_slice_part_infrastructure_monitoring_handle (Slice Part ID): Slice Part Monitoring URL

This API call shall be used to get the management handle of a particular monitoring subsystem that the *Tenant* explicitly requested to be allocated on a Slice Part during the initial Slice instantiation phase. Once the handler (URL) of the monitoring subsystem has been returned, the *Tenant* will be able to use it either to perform further customisation operations (*e.g.*, via a GUI) or to directly consume the flow of measurements coming from the Slice Part. For instance, this could be the case of a Prometheus [PROMETHEUS] instance explicitly allocated for the *Tenant* on a Slice Part, for which the URL endpoint of the related REST interface is provided back as the result of the invocation of this call.

push_handler_to_sro (Service KPI Stream

The *Tenant* usually takes care of the instantiation of the monitoring subsystem that is required for collecting measurements related to the services running on a Slice. The *Tenant* is also the consumer of those measurements, which are utilised to perform service-related orchestration functions. However, in some particular scenarios, it might be required to provide those service-related KPIs also to the Slice Resource Orchestrator in order to implement particular (AI) slice orchestration functionalities (*e.g.*, during the learning phase of a machine learning based orchestration technique).

A *Tenant* willing to explore the usage of these particular (AI) slice orchestration functionalities will provide the SRO with a stream of service-related measurements via the API call **push_handler_to_sro**.

5.2 Cloud-to-Cloud APIs

As soon as a *Tenant* completes the submission of a slice instantiation request to one of the entry points of the NECOS platform (using any of the methods of the *Client-to-Cloud API* described in Section 5.1.1), the involved *NECOS Slice Provider* processes that request and starts the instantiation of the corresponding end-to-end slice.

Considering the interaction workflow described for the API call **create_slice (Service Specification): Slice ID** in Section 5.1.1, when a *Tenant* wishes to deploy the slice via the specification of a (type of) service that should be running on the slice, he should first describe that particular service. This process may involve an interaction with a *Service Marketplace* offering different service parts, however this step is out of the scope of this document. Once the service to be deployed is defined by the *Tenant* using an arbitrary combination of the logical service elements, and after the invocation of the **create_slice** call, a slice descriptor is generated by the *Slice Specifications Processor* and sent to the *Slice Builder*.

A similar (simplified) workflow is also executed when the *Tenant* submits the request for a slice instantiation via the **create_slice (Slice Specification)** or **create_slice (Slice Requirements)** API calls, described in Section 5.1.1. A Slice descriptor is eventually generated also in this case, and provided again as input to the *Slice Builder* (by the *Slice Specifications Processor*) to actually start off the slice instantiation process via the *Cloud-to-Cloud API*.

The *NECOS Cloud-to-Cloud* API consists of 4 different interfaces as depicted in Figure 35, *i.e.*, the *Slice Request Interface*, the *Slice Instantiation Interface*, the *Slice Marketplace Interface*, and the *Slice Runtime Interface*. Details related to each of the above interfaces will be provided in the remainder of this Section.

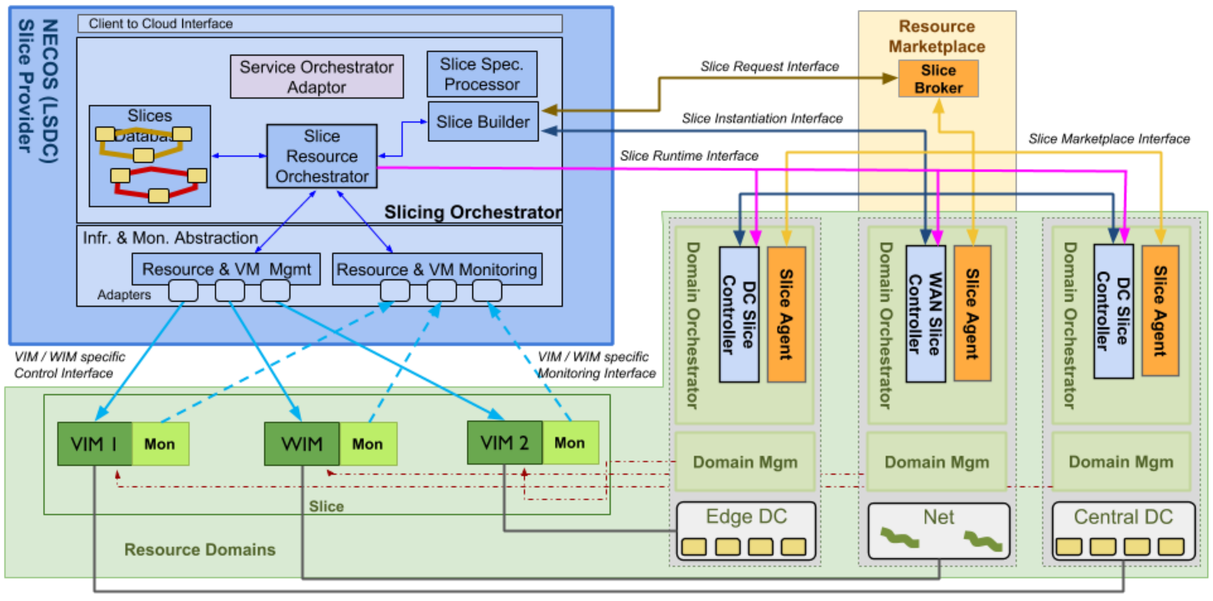


Figure 35. NECOS Cloud-to-Cloud API.

5.2.1 Slice Request Interface

This API initiates the slice creation process. We assume that the *Slice Builder*, after receiving the invocation of the **initiate_slice_creation** call on its internal interface (this API has not fully been detailed in the current release of the document and will be further described in the final version of this deliverable), interacts with the *Slice Broker* (in the *Resource Marketplace*) to request an updated view of the resources available from the different providers that have “registered” their availability to that *Slice Broker*. The term “registered” is used here to identify whatever form of interaction between the

Slice Broker and the underlying *Slice Agents* (the communication mechanisms are not restricted to any particular technology).

The *Slice Request Interface* should provide the methods described in the following in order to support the functionalities required by the *Slice Builder* to discover and select resources made available from (external) resource *Providers* that can be used to build the requested end-to-end slice.

locate_slice_resources (Partially Defined Template): Service Resource Alternatives

The *Slice Builder* starts the slice instantiation process by invoking the **locate_slice_resources** API call (on the *Slice Broker*). The method requires the *Tenant* input in the form of a **Partially Defined Template** (PDT), which covers both the Slice Descriptor and the Slice Requirements. The *Broker* first identifies which *Providers* are able to supply resources for the required slice parts, according to the overall view that the *Slice Broker* collected via the *Slice Marketplace Interface*. The *Broker* responds with a **Service Resource Alternatives** (SRA) message which includes a list of the Slice Controllers corresponding to the providers offering resources.

5.2.2 Slice Instantiation Interface

The next phase of the slice creation process involves interactions via the *Slice Instantiation Interface* between the *Slice Builder* and the set of *Slice Controllers* retrieved earlier via the *Slice Request Interface* (*i.e.*, after the invocation of the **locate_slice_resources** call). Based on the list of entry points of the *Slice Controllers* and on the slice topology partitioning returned by the *Slice Broker*, the *Slice Builder* submits individual requests to the relevant *Slice Controllers* to allocate resources for each single part of the end-to-end slice. The descriptors and requirements for each slice part are being produced based on the PDT message.

The *Slice Instantiation Interface* provides the following methods in order to support the functionalities required by the *Slice Controllers* to reserve, activate, and release the elements of a slice part.

request_slice_part (Slice Part Descriptor, Slice Part Requirements): Slice Part ID

This API method is exposed by the *Slice Controllers*. The **Slice Part Descriptor** (which is generated based on the Slice Specification or the Service Specification) details the characteristics of either a DC or network part of the overall slice, whereas the Slice Part Requirements (which are generated from the Slice Requirements) are used to provide information about the performance requirements for that particular slice part, as derived from the initial description of the end-to-end Slice. Resources are reserved on the *Slice Controller* that receives the invocation of this API call, and a **Slice Part ID** is returned back as a response, in case of a successful interaction.

activate_slice_part (Slice Part ID): {Slice Part ID, Infrastructure Manager Handle}

This API method is also part of the interface exposed by the *Slice Controllers*. This particular method is used to actually activate a slice part (identified by the **Slice Part ID**) on that *Slice Controller* and deploy the on-demand VIM/WIM (Slicing Mode 0) or a shim object on behalf of the tenant (Slicing Mode 1). As a result of the call, a handle to the relevant Infrastructure Manager (*e.g.*, a VIM / WIM) or shim object that was allocated in the Slice Part is returned.

delete_slice_part (Slice Part ID)

This method is exposed by the *Slice Controller* to allow the deletion of a slice part. As a result of the call, the allocated resources/elements of the slice part are released.

5.2.3 Slice Marketplace Interface

This API is related to the interaction between the *Slice Broker* and the *Slice Agents* to implement mechanisms for the propagation of resource offerings between (external) resource domains. As already discussed earlier, different *Slice Agents* register their resources' availability to the *Slice Broker*. The interaction involving the *Slice Broker* and the underlying *Slice Agents* (*i.e.*, communication mechanisms and protocols) are pluggable and not constrained to any specific implementation / technology. The API is aligned to the *NECOS Marketplace* detailed in Section 4.

register_provider (Agent Entry Point): Provider ID, Location

This API method is exposed by the *Slice Broker* in order to select candidate providers. The Agent Entry Point describes the *Providers*' *Slice Agents*, which announce their presence to the *Slice Broker* and then interact to provide offers regarding their resources. Every *Provider*, either *DC* or *WAN*, returns to this API call its ***Provider ID*** along with its geographic ***Location***. The former information is used by the *Broker* when responding to the *Builder* by loading the appropriate fields in the Service Resource Alternatives (SRA) message, *e.g.*, ***dc-slice-controller***, ***dc-slice-point-of-presence*** (*i.e.*, response to the corresponding **PDT** message). The latter is evaluated to meet the *Tenant's* geographic slice constraints.

push_resource_offer (Resource Descriptor): Resource Element ID [], Cost

This API method is also exposed by the *Slice Broker* in order to receive a pool of offers regarding either a DC or a network part of a whole slice. The **Resource Descriptor** field specifies the kind of resource to be offered, while the **Resource Element ID** list sent by the *Slice Agent* is explicitly defining the attributes of a specific piece of resource element, *e.g.*, a Host. For example, while the **Resource Descriptor** indicates the offer of a ***dc-vdu***, the **Resource Element ID** list sent by the *Slice Agent* contains the full list of attributes corresponding to a specific ***host-epa***. The resource offer is accompanied by its corresponding **Cost** and takes part in the formation of a Service Resource Alternatives (SRA) message.

pull_resource_offer (Slice Description): Resource Element ID [], Cost

A similar API to the **push_resource_offer** is the **pull_resource_offer** API which, however, is exposed by the *Slice Agents* and returns **Resource Element ID** lists and their **Cost** as a response to a **Slice Description** request submitted by the *Slice Broker*. More specifically, once the *Broker* receives the **PDT** message, it decomposes it in different queries towards the *Slice Agents*. Each query specifies the **Slice Description** of a slice component, which defines the resource specifications of the latter. The **Slice Description** is included and further defined in the **Resource Element ID** list returned by the *Slice Agent*. In practice, while the ***dc-vdu*** fields define a set of preferences regarding the specifications of a ***host-epa***, *e.g.*, **storage_gb: {in_range: [2, 4]}**, the same fields are explicitly defined inside the *Slice Agent* response, which is the response of this API call, *e.g.*, ***storage_gb: 4***. The responses are used in the Service Resource Alternatives (SRA) message, which is the response of the *Slice Broker* to the *Slice Builder*.

5.2.4 Slice Runtime Interface

This API implements the interface that provides functionalities to dynamically modify the resource allocation for a slice part. This is required by the *Slice Resource Orchestration* to perform lifecycle operation on the end-to-end slice according to the feedback received by each slice part via the monitoring measurements.

get_slice_part_elements (Slice Part ID): Slice Part Element ID []

This API method is exposed by the *Slice Controllers* to allow the *Slice Resource Orchestrator* to get the references to the elements of a given slice part. When invoking this call using a given **Slice Part ID**, a list of **Element IDs** for that slice part is returned to the *Slice Resource Orchestration* that invoked the call.

get_element_handle (Element ID): Element Management URL

This API method is exposed by the *Slice Controllers* to allow a *Slice Resource Orchestration* to retrieve a proper Management entry point associated to the element of a slice part, identified by the **Element ID**.

add_element (Slice Part ID, Element Specification)

This API method is exposed by the *Slice Controllers* to allow a *Slice Resource Orchestrator* to dynamically add a new element into an already allocated slice part. Upon receiving this call, the *Slice Controller* looks for an element matching the element specification and adds that element to the slice part.

delete_element (Element ID)

This API method is exposed by the *Slice Controllers* to allow a *Slice Resource Orchestrator* to dynamically remove an element from an already allocated slice part.

Note that the API methods **get_slice_part_elements**, **add_element** and **delete_element** can be invoked by the *Slice Resource Orchestrator* to provide low-level details of the slice-part elements (and fulfil the tenant's requests described in Section 5.1.2.1), as well as to provide high-level information of the slice part (and fulfil the tenant's requests described in Section 5.1.2.2). In the latter case, the *Slice Resource Orchestrator* may derive the required high-level information from the low-level one.

update_slice (Slice Part ID, Slice Part Descriptor)

A *Slice Resource Orchestrator* that requested the allocation of a slice part and received the associated **Slice Part ID** after its successful instantiation, is able to modify the elements of the slice part by providing an updated Specification for it (via the Slice Part Descriptor parameter). The API call is being processed by the *Slice Controller* that adjusts the arrangement of the slice elements according to the delta between the existing and the new Slice Part Descriptor.

6 Conclusions

This deliverable documents the final version of the NECOS Marketplace, including all associated methods and workflows for resource discovery and matching, as well as a new information model with inherent support for network slicing. The resource discovery and matching processes are applicable to a multi-provider setting, at which requested resources are matched against various resource offerings from multiple infrastructure providers. The associated cost model enables the computation of the cost of slice parts and slice instantiations out of individual (*e.g.,* compute, network, storage) resource prices, providing the means for cost-effective slice deployments when combined with resource selection mechanisms (*e.g.,* exact or heuristic methods). Furthermore, the resource discovery methods support vertical and horizontal slice elasticity. All the Marketplace related methods (*i.e.,* resource discovery, matching) are exemplified through comprehensive and detailed YAML descriptions.

The NECOS information model, being one of the most notable artefacts, combines all necessary resource information to meet the demands of tenants, infrastructure providers, and network slicing orchestrators when dealing with the instantiation, run-time management, configuration, and monitoring of slices and services deployed therein. The information allows each stakeholder to access resource information at the desired granularity, facilitating all related slice operations. To this end, the model enables three main viewpoints (*i.e., Tenant*, *Slice Database,* and *Infrastructure Description).*

The deliverable provides detailed descriptions of all supported cloud APIs, which are either invoked by the *Tenant* during slice request, configuration, run-time management (*i.e.,* client-to-cloud APIs) or by the NECOS orchestrator for slice provisioning, control, and resource orchestration (*i.e.,* cloud-to-cloud APIs). The cloud APIs also include methods for interfacing with external monitoring subsystems, facilitating the run-time operation and SLA management of services deployed within network slices. All these specifications have been made after careful inspection of the SOTA, which has been extensively analyzed in this deliverable.

Despite all supported methods, operations, and models of the Marketplace, the provisioning and run-time management entails additional challenges, many of which go beyond of the scope of this project, and, as such, may be worth investigating in future work:

- **Multi-provider slice embedding.** Optimizing the embedding of slices among different infrastructure providers comprises a very challenging problem, due from to strict provider policies (with respect to information disclosure) and tussles between contrasting provider and client utility functions [DIETRICH2017].
- **Resource management with deep learning.** Resource management in network slicing environments raises the need for online decision making, which, in turn, require a profound understanding of the specific workload and the environment. Deep reinforcement learning can provide the means for the development of techniques that learn to manage resources directly from experience [LIBERATI].
- **Cross-slice communication.** Network slicing is commonly mandated by strong isolation, essentially preventing any cross-slice communication and access between service components even if they are placed on the same datacenter or server. Facilitating and orchestrating the direct communication between the corresponding service components, within the boundaries of the same datacenter or server, can yield significant latency benefits, as well as traffic savings both in the backhaul/transport segments of the network [MESON, KATSAROS2019].
- **Network slice programmability.** Granting advanced control to the tenant for the programming of the network elements of his slice (*e.g.,* allowing him to implement custom forwarding decisions on his own) would significantly provider interventions during the run-time operation of a slice [BOZAKOV]. Software-defined networking (SDN) and recent network programming languages can provide the basic means for building systems that enable a higher degree of slice network programmability.
- **Marketplace KPIs.** Extending the slicing KPIs identified in [3GPP-KPIs] to the Resource Marketplace and utilising them in the various Marketplace operations.

7 Appendix

In addition to Section 2, this appendix provides a detailed state-of-the-art analysis in terms of cloud APIs and information models for service and resource descriptions. In particular, Section 7.1 discusses relevant cloud APIs, whereas Section 7.2 describes relevant information models.

7.1 Cloud APIs

7.1.1 5G-PPP

The 5G-PPP white paper, entitled "*View on 5G Architecture*", proposes the 5G architecture illustrated in Figure 36, as a result of the composition of various individual 5G-PPP initiatives and research projects. On this basis, the following functionalities of the architecture are analyzed: network slicing, programmability and softwarization, management and orchestration, 5G security, and RAN architecture.

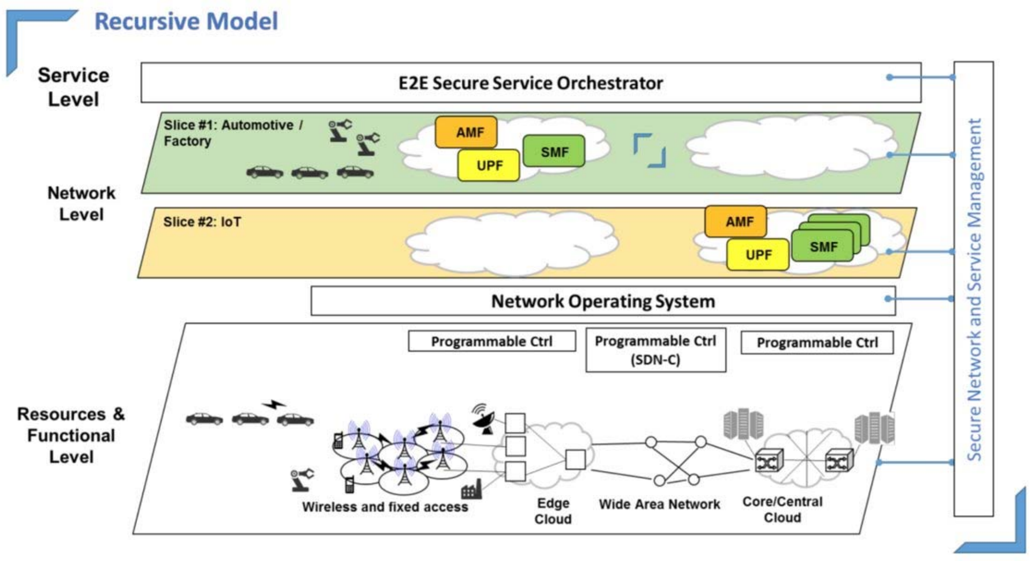

Recursive Model
Service Level
E2E Secure Service Orchestrator
Slice #1: Automotive / Factory
AMF
UPF
SMF
Network Level
Slice #2: IoT
AMF
UPF
SMF
Network Operating System
Programmable Ctrl
Programmable Ctrl (SDN-C)
Programmable Ctrl
Resources & Functional Level
Wireless and fixed access
Edge Cloud
Wide Area Network
Core/Central Cloud
Secure Network and Service Management

Figure 36. 5G overall architecture
(Source: 5G-PPP, View on 5G Architecture v2.0).

For the overall architecture, a recursive structure is proposed, defined as "the ability to build a service out of existing services". In a network slicing point of view, this capability allows a slice instance operating on top of the infrastructure resources provided by the slice instance below. The tenant can operate its virtual infrastructure as it operates the physical one, allocating and reselling part of the resources to other tenants in a recursive manner. As such, each tenant can own and deploy its own MANO system. To provide support for this key functionality, a set of homogeneous APIs are needed to provide a layer of abstraction for the management of each slice and controlling the underlying virtual resources.

The structure of APIs are not on the main objectives of the white paper. However, in a conceptual framework, APIs functionalities are considered in architectures related to the NECOS project and more specifically, network slicing, and management and orchestration architectures.

In terms of network slicing, 5G-PPP proposes a set of APIs for the interaction between Network Services (NS) and the corresponding VNFs that encompass the following attributes: network-slice ID, nodes, links, connections points, storage resources, compute resources, topologies, network services, service-specific managers, network functions, virtual network functions, network function-specific managers and predefined function blocks. Moreover, these information elements are currently under standardisation in the ETSI NFV ISG, in OASIS TOSCA standards, and in IETF. As network slicing services can be grouped to two different levels, *i.e.*, (i) the provisioning of Virtual Infrastructures (VI) and (ii) the provisioning of tenants owned NS, 5G-PPP also provides a general categorization of APIs needed to enable both services providing to different degree of control of network slices, defined as follows:

- Network Service Allocation / Modification / De-allocation API,
- Virtual Infrastructure Allocation /Modification / De-allocation API,
- Virtual infrastructure control API with limited control, and,
- Virtual infrastructure control API with full control.

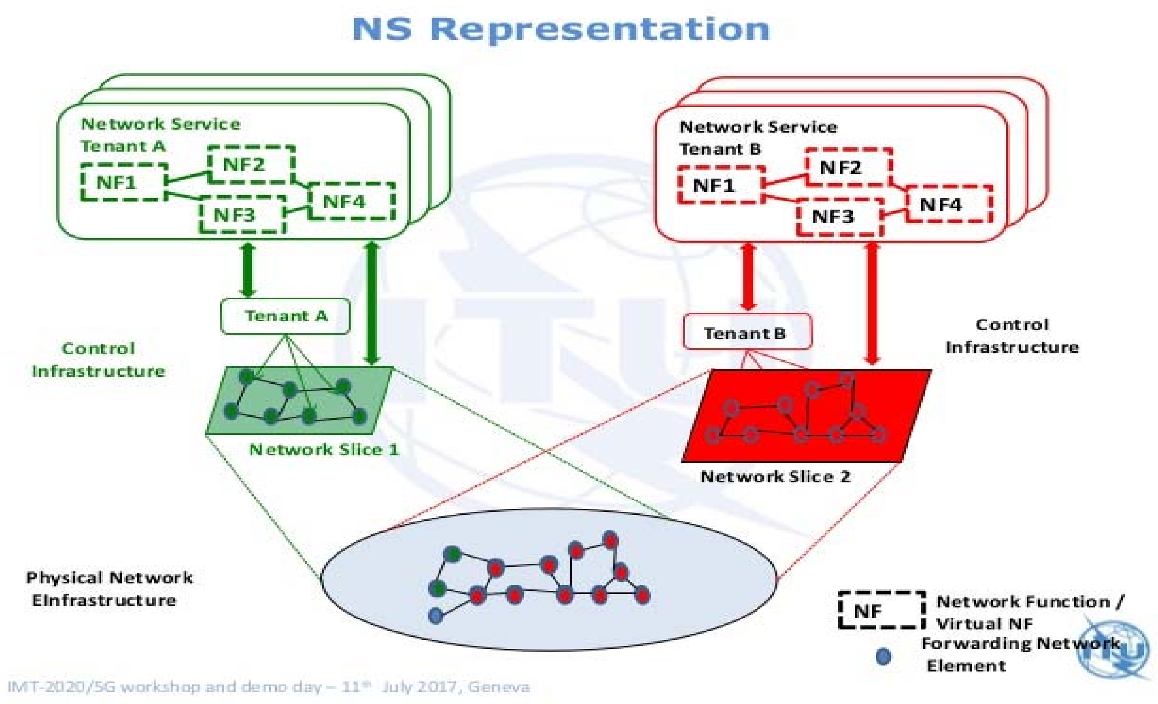


Figure 37. Network service representation
(Source: 5G-PPP, View on 5G Architecture v2.0).

A set of APIs is also required for the multi-domain orchestration, which includes the automated management of services and resources in multi-technology environments (multiple domains involving different cloud and networking technologies) and multi-operator environments (multiple administrative domains). This challenging plane, consisting of various concepts as depicted in Figure 37, has to be supported by several APIs.

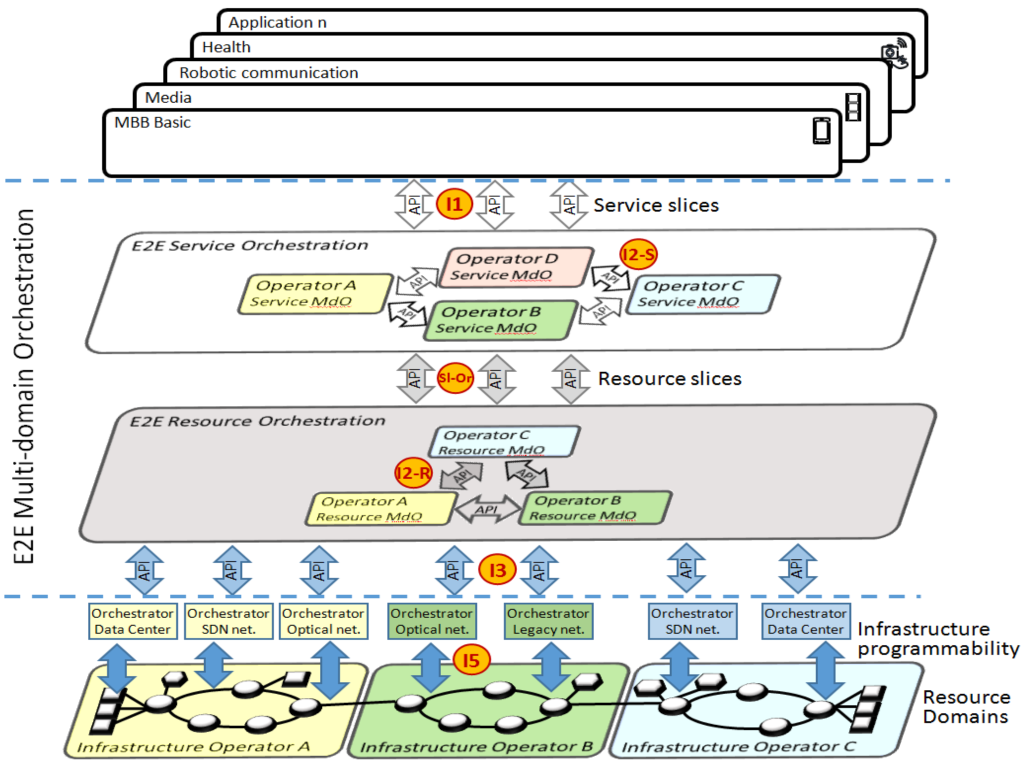


Figure 38. 5G-PPP APIs.

At the lower plane of Figure 38 there are resource domains, exposing resource abstraction on interface I5. Domain orchestrators perform resource orchestration and/or service orchestration exploiting the abstractions exposed on I5 by resource domains.

At Multi-domain orchestrator (MdO) plane, the resource MdO belonging to an infrastructure operator, for instance operator A, interacts with domain orchestrators, via interface I3 APIs, to orchestrate resources within the same administrative domains. The MdO interacts with other MdOs via interface I2-R APIs (business-to-business or “B2B”) to request and orchestrate resources across administrative domains. Resources are exposed at the service orchestration level on interface Sl-Or to Service MdOs. Interface I2-S (B2B) is used by Service MdOs to orchestrate services across administrative domains.

Finally, the Service MdOs expose, on interface I1, service specification APIs (Customer-to-Business or “C2B”) that allow business customers to specify their requirements for a service. The framework also considers MdO service providers, such as Operator D in Figure 38, which do not own resource domains but operate a multi-domain orchestrator to trade resources and services.

To sum up, 5G architecture enables new business opportunities meeting the requirements of a wide range of use cases, as well as enables 5G to be future proof by means of: (i) implementing network slicing in a cost-effective way, (ii) addressing both end-user and operational services, (iii) supporting softwarization natively, (iv) integrating communication and computation, and (v) integrating heterogeneous technologies (including fixed and wireless technologies).

7.1.2 5GEx

The 5GEx architecture framework, shown in Figure 39, identifies the main functional components and the interworking interfaces involved in multi-domain orchestration.

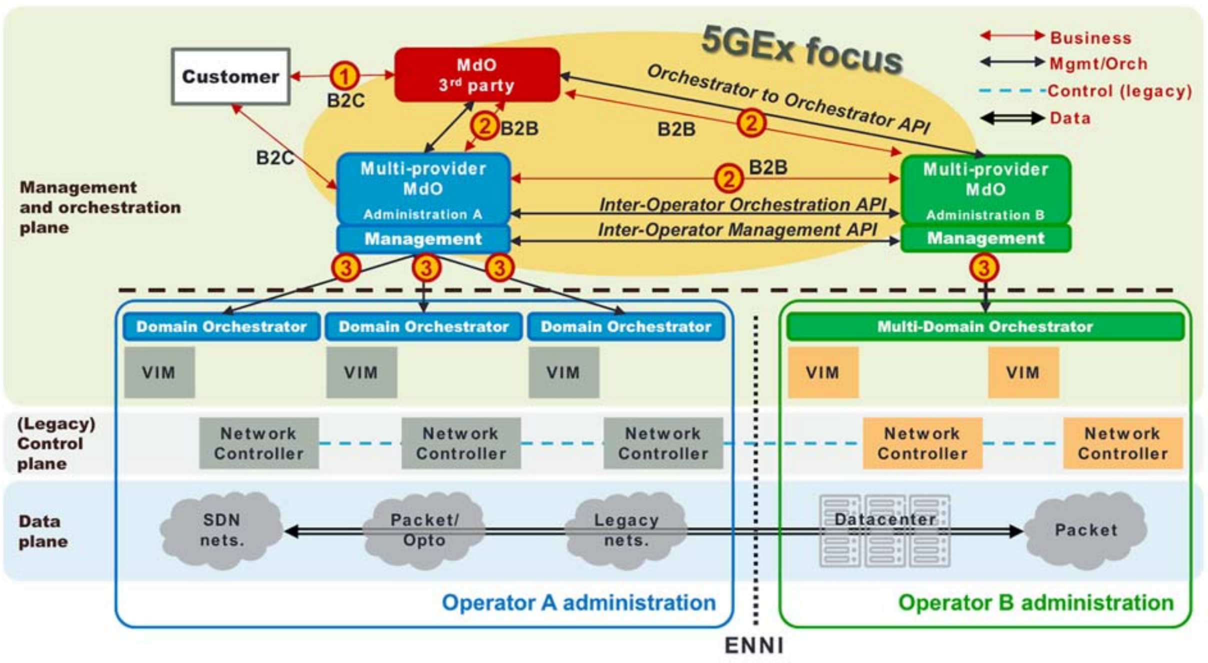

5GEx focus
Business
Mgmt/Orch
Control (legacy)
Data
Customer
MdO 3rd party
B2C
B2B
Orchestrator to Orchestrator API
Multi-provider MdO Administration A
Management
Multi-provider MdO Administration B
Inter-Operator Orchestration API
Inter-Operator Management API
Management and orchestration plane
Domain Orchestrator
Multi-Domain Orchestrator
VIM
(Legacy) Control plane
Network Controller
Data plane
SDN nets.
Packet/ Opto
Legacy nets.
Datacenter
Packet
Operator A administration
Operator B administration
ENNI

Figure 39. 5GEx architecture reference framework
(Source: 5GEx Deliverable D2.2).

The bottom part of Figure 39 shows different Resource Domains, hosting the actual resources. The middle part shows the Domain Orchestrators that are responsible for performing Virtualization Service Orchestration and/or Resource Orchestration exploiting the abstractions exposed by the lower Resource Domains. The key 5GEx component – the Multi-provider Multi-domain Orchestrator (MdO) – is shown at the top of Figure 39. The MdO handles the orchestration of resources and services from different providers, coordinating resource and/or service orchestration at multi-domain level, where multi-domain may refer to multi-technology (orchestrating resources and/or services using multiple Domain Orchestrators) or multi-provider (orchestrating resources and/or services using Domain Orchestrators belonging to multiple administrative domains).

There are three main interworking interfaces identified in the 5GEx architecture framework, briefly described next. The MdO exposes service specification APIs (Business-to-Customer, B2C) that allow business customers to specify their requirements for a service on Interface 1. The MdO interacts with other MdOs via Interface 2 APIs (Business-to-Business, B2B) to request and orchestrate resources and services across administrative domains. Finally, the MdO interacts with Domain Orchestrators via Interface 3 APIs to orchestrate resources and services within the same administrative domains.

In 5GEx, the provisioning of multi-domain services involves a series of actions between SPs consisting of 4 steps: (i) the discovery phase, for the distribution and population of the own capabilities, as well as the formation of the entire view of the multi-domain ecosystem by each of the service providers participating on it in the form of service offerings, (ii) the request phase, where the external customers solicit the provision of services, (iii) the fulfilment phase, where the lifecycle management of the required network functions is handled, and the necessary resources are configured and control, and (iv) the assurance phase, where the service environment is monitored and, as a consequence of that, more control and management functions for lifecycle of the VNFs and configuration of resources could be performed for ensuring service levels.

The different actions identified above led to the need for a further splitting of the functionalities that the generic interfaces 1, 2 and 3 should support according to the following list:

- Service management (Ix-S)
- Catalogues (Ix-C)
- VNF lifecycle management (Ix-F)

- Resource / Topology (Ix-RT)
- Resource / Control (Ix-RC)
- Monitoring (Ix-Mon)
- SLA (Ix-SLA)

The x-S interface is used in the 5GEx architecture for requesting services. Those services can be requested by external customers, making use of 1-S interface, or can be requested between MdOs of different administrative domains, making then use of 2-S one.

In principle, no differences are foreseen between I1-S and I2-S variants, then the subsequent analysis is generalized and applied to both cases. To some extent, the capabilities of the Ix-S interface are similar to the ones required by the Ix-C, described next. While the Ix-C interface is mainly devoted to the sharing of information, the Ix-S interface is used for invoking the services as described through such information-sharing process. From that point of view, it was sensible using the same implementation for both interfaces. That is is based on a YANG information model supported by ETSI.

Catalogues

Ix-C covers: (i) I1-C, which is the interface that allows the interaction with the local catalogue subsystem by the provider of the local domain and by the 5GEx customer, and (ii) I2-C, which is the east/west interface that interconnect catalogue subsystems in two different administrative domains in order to exchange the necessary information to build multi-domain services.

I2-C interface is defined as the interface between two different MdOs through which all the information related to their catalogues (containing Network Services and Network Functions - VNFs) is exchanged, connecting the local Catalogue Management module to its homonym in the neighbour domain.

VNF lifecycle management

The I2-F interface is used to communicate lifecycle management dependencies and workflows of Network Service parts or compound VNFs.

There are several inter-provider network scenarios that involve communication between NFVOs that belong to different administrations. The I2-F interface is used to delegate NS and VNF lifecycle management for some selected components of the Network Service to another provider. Delegation of lifecycle management occurs in general during on-boarding of an NSD or a VNFD. This operation, however, may also take place dynamically as part of a specific NS/VNF instantiation.

Resource / Topology

The I2-RT (Resource Topology) interface is used by a MdO to exchange the network topology and resource information with other MdOs. The information collected through I2-RT enables a MdO to: (i) detect the existence of other domains, (ii) learn about the network connectivity between domains, (iii) acquire details about the resources and service capabilities of specific domains, (iv) obtain adequate details about specific domains needed for the placement of VNF in the global infrastructure, (v) set up connectivity between domains, if required, through I2-RC, (vi) orchestrate connectivity between VNFs of different domains through I2-RC.

Resource Control

The Ix-RC interface is used by the MdO to reserve, provision, configure and manage resources through other MdO's. Two kinds of resources were mainly identified: IT resources and Network resources.

For IT domains, the resources are related to:

- vCPU and memory for the compute node, which are often bundled (*e.g.*, small, medium, big in OpenStack),
- storage space and type of storage,
- IT connectivity between VMs, including remote access outside the IT domain.

For network domains related resource include:

- Bandwidth, loss, jitter, delay to characterize the QoS,
- End-points to determine the tail and head of the connectivity,
- Encapsulation of the packets to describe how the connectivity is rendered.

Monitoring

The realisation of the network service assurance in the context of 5GEx requires the design and implementation of proper mechanisms that allow performing on-demand monitoring of the services instantiated and orchestrated in the considered multi-domain, multi-provider scenario.

A first dimension to be considered for this process referred to the Service Level Agreement (SLA) coming with each submitted service request, which may include different conditions to be verified by checking specific values (*e.g.*, metrics, statistics, etc.) that are relevant for each Service Level Objective (SLO). A second dimension to be considered referred to the particular resources implementing a certain service instance, which are selected at the time of service deployment according to the outcome of the resource orchestration algorithms. The latter may also require the collection of measurements to be used as feedback to the service and resource orchestration processes, thus introducing a third dimension of complexity.

During the analysis it became evident the need for having a separation of concerns on the monitoring functionalities. The Monitoring interfaces were indeed decomposed into two separate sub-interfaces named Ix-Mon control and Ix-Mon data, being applied to both I3 and I2 interfaces.

I3-Mon control is the interface which is expected to provide functionalities for both managing probes lifecycle and requesting the collection/storage of measurements coming from resources related to the local running service instances. The purpose of I3-Mon control would then consist in defining a common way of remotely and dynamically controlling and orchestrating the configuration/activation of monitoring probes to collect and storing measurements from all the different resource domains that are involved in the realization of a given service instance.

The final goal of the probes' activation process consists in enabling the collection of relevant measurements to be used by the MdO management functions for the purpose of service assurance, orchestration of services and resources, etc.

This requirement implies that different measurements, coming from different resource domains but related to the same service instance, will somehow have to logically be linked and then conveyed to a common storage repository in the multi-domain orchestrator.

For the definition of this interface the focus was not on considering the particular mechanisms to be used while interacting with the repository for either writing measurement data (southbound part) or querying them (northbound part) as they may vary with the particular storage technology. In the case of I3-Mon data it was more sensible defining a common, agreed data model to be used when measurements are stored and retrieved.

I3-Mon data in fact required the definition of an abstraction between different resource domains acting as producers of monitoring data, and some MdO management functions taking the role of consumers of monitoring data.

SLA

I3-SLA covers: (i) the flow of monitoring information from the monitoring DB to the SLA manager for its evaluation, and (ii) the events communication between the SLA Manager and the Orchestrator.

I3-SLA interface is an internal domain interface between the local SLA Manager and the local monitoring DB, through which the SLA manager retrieves the monitoring information for the KPIs involved in the SLA contracts for the business transactions.

Every running instance has an associated set of KPIs that needs to be evaluated according to the terms of the SLA. Periodically, the SLA manager will contact the monitoring DB looking for monitoring information for each of the KPIs. This process is split into two steps: the SLA Evaluator will contact the SLA Aggregator which then will process the request. If it is a simple KPI evaluation, the petition will be forwarded to the monitoring DB and the information will be sent back to the SLA Evaluator. If we are dealing with a complex KPI, the SLA Aggregator will create as many requests as needed to the monitoring DB to retrieve all the individual samples for each of the simple KPIs that compose the complex one. After that, the information will be aggregated and sent for evaluation [5GEX].

7.1.3 ITU-T

As well as several SDO, the International Telecommunications Union (ITU) has tried to establish a common ground for all future mobile-broadband communications ecosystem actors, coining the term IMT-2020 (International Mobile Telecommunication system - 2020) to embrace all the efforts to provide an international specification for 5G.

The concept of network slicing stands among the novel concepts therein, as a strategy to build efficient and cost-effective infrastructures that can be shared by several services. According to [GALIS2017], a network slice is “a managed group of subsets of resources, network functions / network virtual functions at the data, control, management/orchestration, and service planes at any given time. The behaviour of the network slice is realized via network slice instances (*i.e.*, activated network slices, dynamically and non-disruptively re-provisioned). A network slice is programmable and has the ability to expose its capabilities”.

For ITU-T, network slicing is perceived as Logical Isolated Network Partitions (LINP). According to Recommendation ITU-T Y.3011, a LINP is composed of multiple virtual resources, whose capability may be not bound to the capability of the physical or logical resource, which are isolated and equipped with a programmable control and data plane.

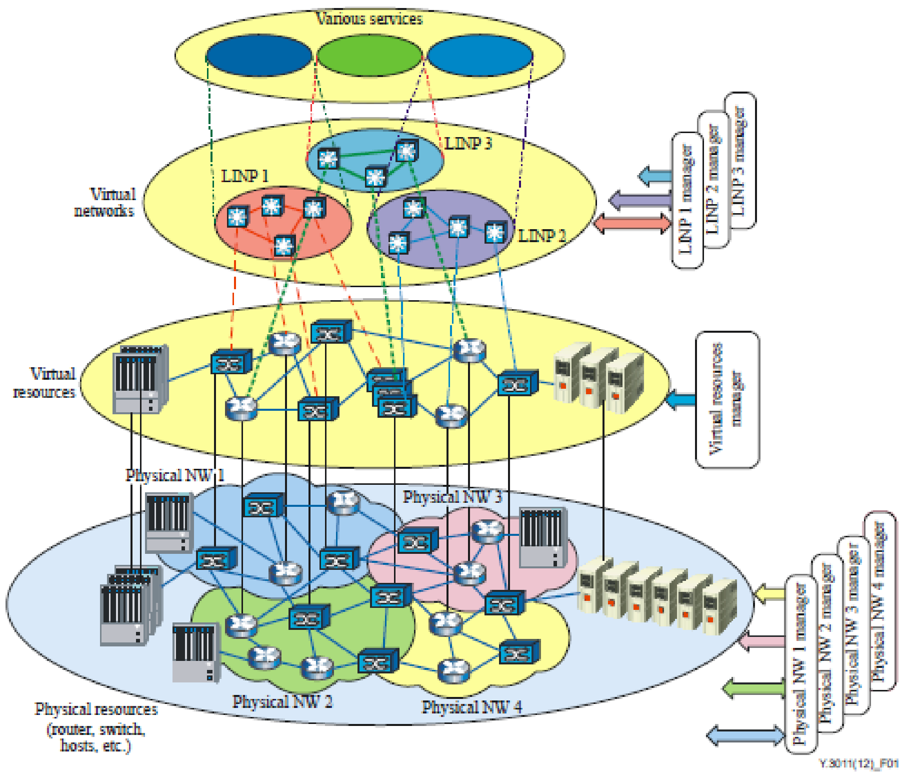


Figure 40. Conceptual architecture of network virtualization (Source: Recommendation ITU-T Y.3011 - 01/2012).

Thus, network virtualization is seen as a method that allows multiple LINPs to coexist in a single physical network. Figure 40 presents the conceptual architecture of network virtualization. It can be seen that a single physical resource can be shared among multiple virtual resources and each LINP consists of multiple virtual resources. Each LINP is managed by an individual LINP manager. Moreover, the physical resources in a physical network(s) are virtualized and may form a virtual-resource pool, which is managed by the virtual resources manager (VRM). The VRM interacts with the physical network manager (PNM) and performs control and management of virtual resources.

Figure 41 depicts the LINP concept and coexistence among a multitude of LINPs, comprising several resources, physical or virtual, to support network virtualization. As presented in Recommendation ITU-T Y.3011 – 01/2012, there shall have a strict relationship between a LINP and user requirements. Such requirements provide a basis for VRMs to coordinate the allocation of appropriate LINPs to a given user/set of users, based on VRM administration policy. Moreover, each LINP is controlled and managed by an LINP manager. The VRM which is controlling all virtual resources creates an LINP manager and allocates appropriate authorities to control each LINP.

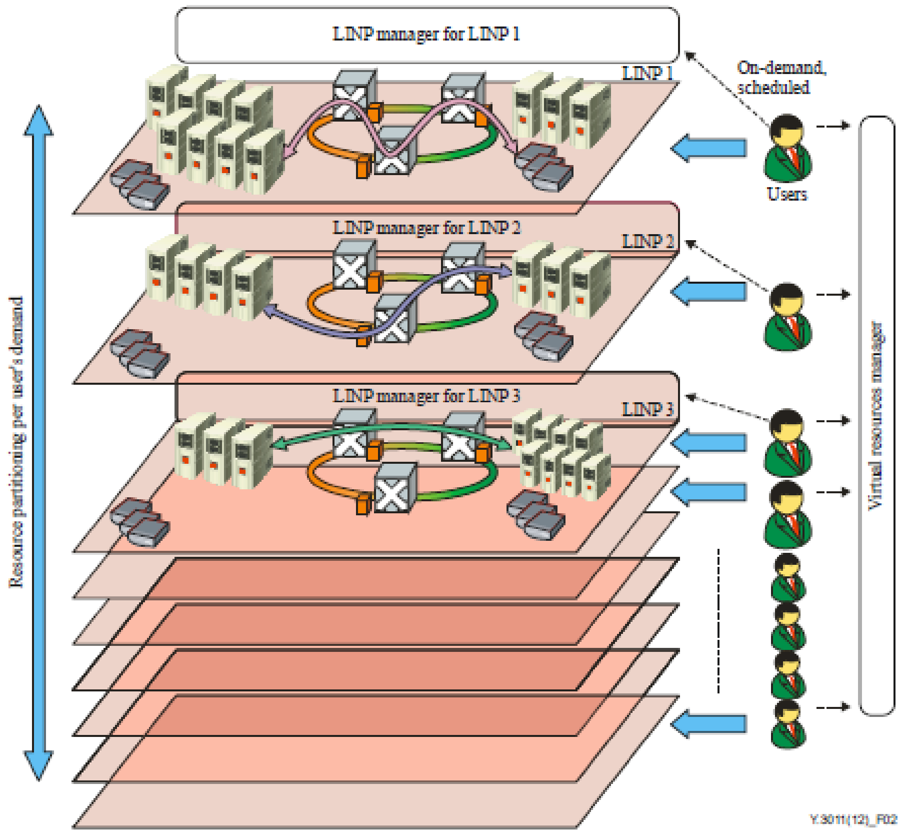


Figure 41. Concept of LINP provided by network virtualization (Source: Recommendation ITU-T Y.3011 – 01/2012).

A LINP generated by network virtualization has various characteristics, such as partitioning, isolation, abstraction, flexibility or elasticity, programmability, authentication, authorization, and accounting. However, to achieve such a set of goals, ITU has identified several missing points, based on a gap analysis cited in [ITU-T Y.3011]:

- Lack of a unified network management structure;
- Non-standardized Operations and management (OAM) protocols;
- Non-existing strategies to manage and orchestrate the softwarized network components, as well as to softwarize network management and orchestration functionality; and
- Lack of a "network slice-driven" lifecycle management and orchestration.

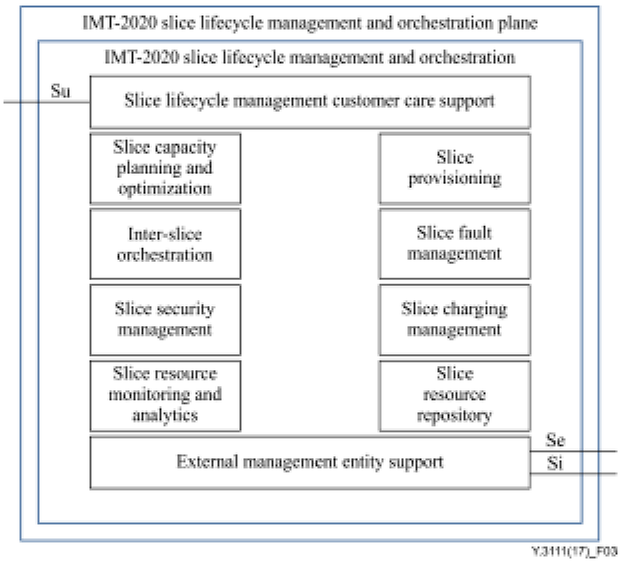

IMT-2020 slice lifecycle management and orchestration plane
IMT-2020 slice lifecycle management and orchestration
Su
Slice lifecycle management customer care support
Slice capacity planning and optimization
Slice provisioning
Inter-slice orchestration
Slice fault management
Slice security management
Slice charging management
Slice resource monitoring and analytics
Slice resource repository
Se
Si
External management entity support
Y.3111(17)_F03

Figure 42. Network slice lifecycle management and orchestration functional components (Source: Recommendation ITU-T Y.3011).

Moreover, it is considered that the management and orchestration architecture in IMT-2020 is also required to deal with two levels: network slice life-cycle management, as well as in each network slice instances – Instances 1 and 2, respectively. Given this architectural approach, Recommendation ITU-T Y.3011 specifies a network management and orchestration framework for IMT-2020 in order to accomplish the goals set above.

According to this ITU recommendation, network slice orchestration functionalities are specified in the functional elements: slice capacity planning and optimization, slice provisioning (SP), and inter-slice orchestration, while the management functionalities are specified in the functional elements slice fault/security/charging management, slice resource monitoring and analytics and resource repository, working together to achieve the slice lifecycle management objectives. Figure 42 shows this set of functional components.

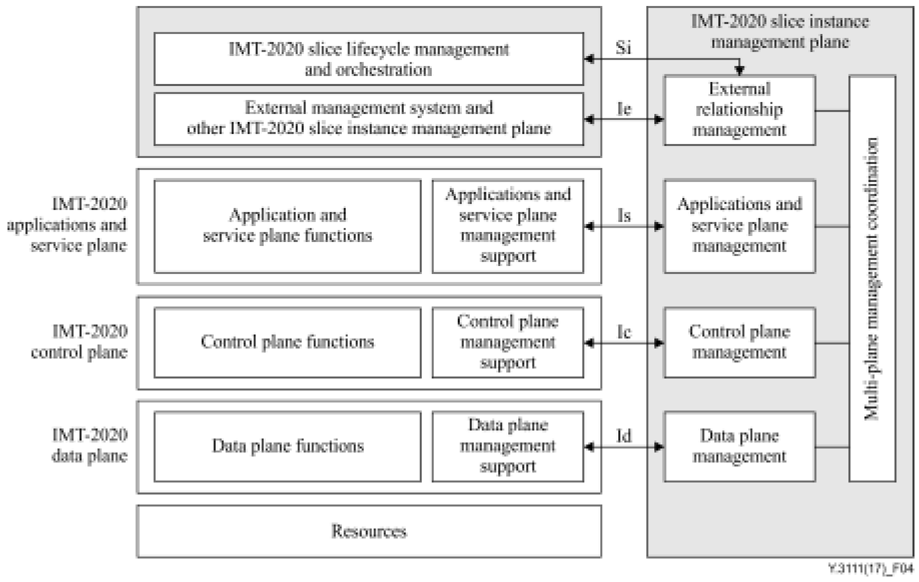

IMT-2020 slice lifecycle management and orchestration
External management system and other IMT-2020 slice instance management plane
Si
Ie
IMT-2020 slice instance management plane
External relationship management
IMT-2020 applications and service plane
Application and service plane functions
Applications and service plane management support
Is
Applications and service plane management
IMT-2020 control plane
Control plane functions
Control plane management support
Ic
Control plane management
IMT-2020 data plane
Data plane functions
Data plane management support
Id
Data plane management
Multi-plane management coordination
Resources
Y.3111(17)_F04

Figure 43. Network slice instance management functional architecture (Source: Recommendation ITU-T Y.3011).

Similarly, ITU-T Y.3011 also specifies the network slice instance management functional architecture and components, as well as its relationships and interfaces with slice lifecycle management and orchestration functional component and external management systems, as illustrated in Figure 43.

Recommendation ITU-T Y.3011 Section 11 presents the slice lifecycle management procedure, as stated below in brief:

- The IMT-2020 customer requests a slice to be provisioned with its specified service requirements;
- IMT-2020 slice lifecycle customer care support (SLMCCS) functional element receives the customer's request and carries it to the slice capacity planning and optimization functional element (SCPO). SCPO then determines an optimal slice plan based on the available resources which match the customer's request;
- Once the provisioning policy is determined, SCPO requests provisioning to slice provisioning (SP) functional element. SP then performs the requested slice provisioning task. Upon completion of the provisioning process, SP sends a provision reply message to the customer via SLMCCS. At the same time, it sends a provision status slice resource monitoring and analytics functional element to initiate the collection and monitoring of the provisioned resources. It also sends the status update to slice resource repository (SRR) to store the provisioned resource information;
- Slice Resource Monitoring and Analytics (SRMA) performs collection, monitoring, and analysis tasks of the provisioned slice resources. Data and information collected and analysed is then stored in SRR for further processing by other functional elements;
- When SRR receives any resource status updates, it stores them in the repository and, at the same time, it emits notification to all functional elements that are listening to the status updates (slice fault management - SFM, slice security management - SSM, slice charging management - SCM, and SCPO);
- When SCPO receives the notification, it updates available resource status and determines if re-optimization is needed upon status updates;
- SP, upon receiving the provisioning update requests, performs re-provisioning tasks for the provisioned slices, generating provision status reports to the related functional elements.

In April 2018, ITU-T has released Draft Recommendation ITU-T Y.3112 (Y.IMT2020-MultiSL) - Framework for the support of Multiple Network Slicing. As presented in its summary, this Recommendation describes the concept of network slicing and use cases of multiple network slicing, enabling a single device to simultaneously connect to different network slices. The use case describes the slice service type for indicating a specific network slice and the slice user group for precisely representing the network slice in terms of performance requirements and business models. Finally, it also specifies the high-level requirements and high-level architecture for multiple network slicing in IMT-2020 network.

As far as it can be seen, APIs for network slicing in clouds is still a pending issue for ITU-T IMT-2020 future evolvements.

7.1.4 Open Grid Forum

The Open Grid Forum (OGF) is committed to the evolution and adoption of advanced applied distributed computing, such as cloud, grid, and networking, through a highly involved open community. The OGF aims at developing and promoting innovative scalable techniques, applications, and infrastructures in order to increase productivity in both enterprise and academy communities. The open community consists of thousands of individuals spread out in industry and research, representing more than 50 countries, over 400 organizations.

The work is carried out through community-initiated working groups that collaboratively develop standards and specifications with other leading standards organizations, software companies, and future users. Its organizational members, including technology companies and research institutions in academia and government, are responsible for funding the OGF. Several events to further develop grid-related specifications and use cases are hosted by OGF each year.

Currently, the OGF working groups have been studying several proposals, and the most relevant to NECOS is the Open Cloud Computing Interface (OCCI), which focuses on the cloud computing IaaS based model. OCCI is a protocol and API that aims to enable the development of interoperable tools for common tasks, including deployment, autonomous scaling, and monitoring.

As shown in Figure 44, the OCCI interface is a boundary protocol and API that acts as a service front-end to a provider's internal management framework, and it is placed in a provider's architecture.

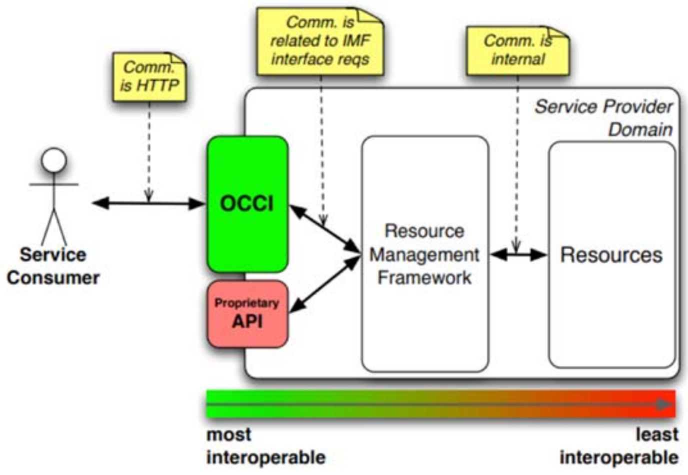


Figure 44. OCCI Interface (Source: OCCI, http://occi-wg.org/about).

End-users and other system instances can be seen as service consumers, and OCCI is suitable for both cases. As a key feature, it can be used as a management API for all kinds of resources, while at the same time maintaining a high level of interoperability.

In summary, OCCI is able to abstract and generalize methods or call specific functions of a particular VIM (or any other management software). It does not have an API to directly instantiate or monitor a slice. Some close features can be found at the OpenStack OCCI Interface implementation (https://github.com/openstack/ooi), as shown in Figure 45, which is capable of invoking OpenStack standard commands in a generic way.

In a general way, with the exception of OCCI, OGF does not have standards that may contribute to the specification of both Client-to-Cloud and Cloud-to-Cloud APIs. That is, the OGF still has no efforts aimed at slicing or networking slice. There are OCCI implementations that leverage communication with some VIMs, such as OpenStack, OpenNebula and CloudStack; but with a very specific focus on cloud computing.

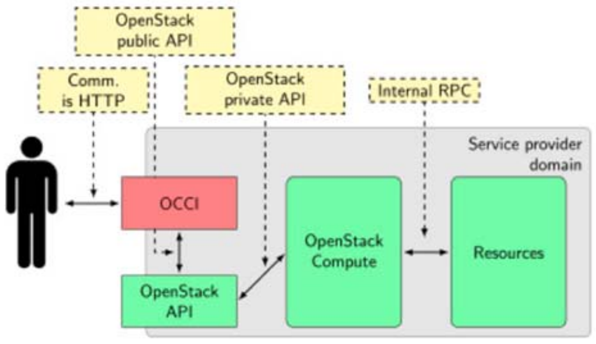


Figure 45. Openstack OCCI interface implementation
(Source: OCCI, http://occi-wg.org/tag/openstack).

7.1.5 Initiatives in Other Standards Development Organizations

Some other Standards Development Organizations (SDOs), as well as industrial associations are looking at the network slice concept from different angles and perspectives [CONTRERAS18]. From the provider's point of view, there is a risk of fragmenting the conceptual approach to network slices, since small differences can provoke incompatibilities among the different approaches. It is, therefore, necessary to reach consensus on common terms, definitions, rationale, ideas, and goals to properly normalize the concept of network slicing.

The NGMN Alliance [NGMN] has provided a primary description of the network slice concept as mentioned in the introductory section. The NGMN view is that of a 5G slice as a composition of a collection of 5G network functions and specific Radio Access Technology settings that are combined for the specific use case or business model, while leveraging NFV and SDN concepts. The network slice concept is organized in a layered manner [NGMN], differentiating the service instance layer, comprising the end-user of business services; the network slice instance (NSI) layer, as a set of functions forming a complete instantiated logical network; and the Resource layer, consisting of both physical and logical resources. In this layered view, the NSIs can be potentially shared among multiple service instances.

3GPP [3GPP] differentiates among network slices and network slice instances. On one hand, a network slice represents a logical network providing specific network capabilities and network characteristics. On the other hand, a network slice instance is defined as a deployed network slice, that is, a specific set of network function instances and associated resources.

ETSI NFV [NFV] specifies network operators' perspectives on NFV priorities for 5G, network slicing support with ETSI NFV architecture and an E2E network slicing framework. Another recent development within ETSI Zero Touch Network and Service Management Industry Specification Group (ZSM ISG) is specifically devoted to the standardization of automation technology for network slice management [GOTO18]. Within the ETSI Multi-access Edge Computing (MEC) group, a new work item called "MEC support for network slicing" [MEC] seeks to identify the necessary support for network slicing, evaluating the gaps from MEC features and functions, and identify the new requirements.

The BBF [BBF] is also approaching network slicing by augmenting the previous management functions by defining new and complementary ones, such as Access Network Slice Management, Core Network Slice Management, and Transport Network Slice Management. Each one of them is intended to take care of the slice lifecycle management of each particular network slice subinstance (*i.e.*, access, core, or transport).

7.2 Information models

7.2.1 NOVI

The NOVI project [NOVI2015] defined an architecture for supporting federation of infrastructures. NOVI has a Service Layer that allows users to have a unique interface to access and use resources in different testbeds. Access to the testbeds, authorization policies, monitoring information and selection of resources are integrated among platforms and implemented in the NOVI layer. The project identified the definition of a common Information Model (IM) as the essential element to achieve the federation goals. Their model was intended to support virtualized resources and context-aware resource selection; to be vendor-independent; to support monitoring and measurement concepts, and to support management policies.

NOVI uses Web Ontology Language (OWL) for modelling virtualization explicitly for both, computing and networking devices using Web Ontology Language (OWL). It is vendor-agnostic, modular, and composed of three main ontologies: (i) resource ontology, (ii) monitoring ontology and (iii) policy ontology.

Resource Ontology

Figure 46 partially depicts NOVI's resource ontology. Besides the ontology-based substrate, it is a more or less common representation of resources and services.

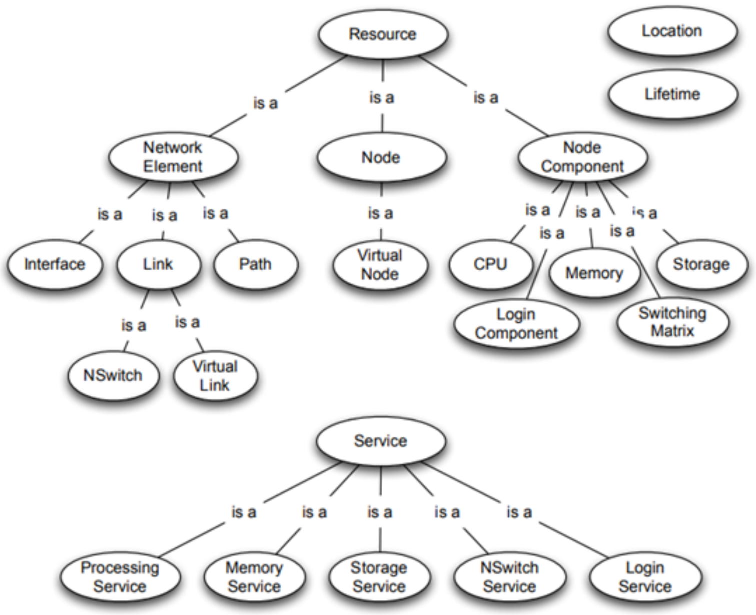


Figure 46. NOVI ontology for modelling resources and services (Source: [NOVI2015]).

In particular, it is worth noting the classes "Location" and "Lifetime". The former is an approximate geographical location to describe which resources share the same location. It can also be extended with properties such as GPS coordinates. The latter is used to describe the time dimension of a reservation, but it can also be used to describe the availability of nodes, *e.g.*, that a node is not available during a maintenance period.

The Service class allows the user to express the desired service-level. This allows the user to decouple the service request from the actual physical implementation.

Figure 47 shows how network elements are connected in paths and nodes with unidirectional links. The authors justify this decision saying that “has been a conscious choice to follow the Network Markup Language (NML) [NML] model, which follows the philosophy that a unidirectional model can describe a bidirectional model, but not vice versa”.

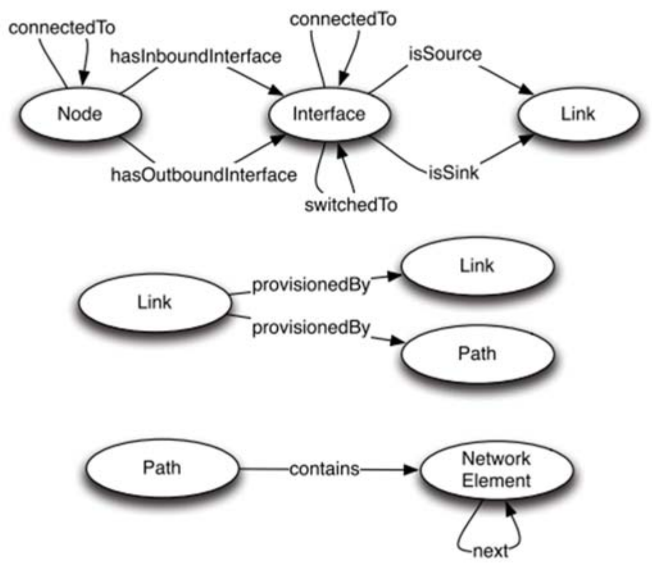


Figure 47. Network connectivity properties defined in the NOVI resource ontology (Source: [NOVI2015]).

Monitoring Ontology

Figure 48 depicts NOVI's Monitoring Ontology. An interesting aspect is the inclusion of modules, such as the Query model or the Statistic Model, which are semantic bridges between data consumers and data generators.

This model is designed to store static and dynamic information. By static information, NOVI refers to constant characteristics of the resource, or those that may change very infrequently (*e.g.*, number of CPUs in a server). Dynamic information encompasses attributes, such as the utilization of a CPU core. In NOVI, the description of resources is carried out at two abstraction levels: physical resources, and virtual resources. At the physical resource level, when the user requests a virtual testbed (a Topology), it may contain runtime, dynamical constraints, such as CPU or main memory. At the virtual level instead, there is the monitoring support for the virtual testbed. Given that the user successfully acquires a virtual topology, NOVI offers services to keep track of its certain temporal variables. For instance, a user is interested in the evolution of the round trip delay in certain links of his topology. These two complementary abstraction levels split between substrate monitoring and slice monitoring.

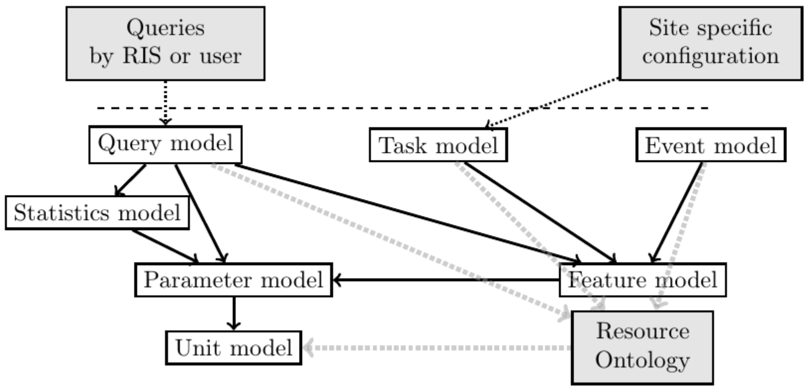


Figure 48. NOVI's modular ontology for modelling monitoring data and tasks (Source: [NOVI2015]).

It is also interesting to see the existence of the unit model where the fundamental concepts of the Monitoring Ontology are laid down; these are definitions of levels, dimensions, units and unit prefixes. This yields clear semantics to the data stored in the monitoring ontology.

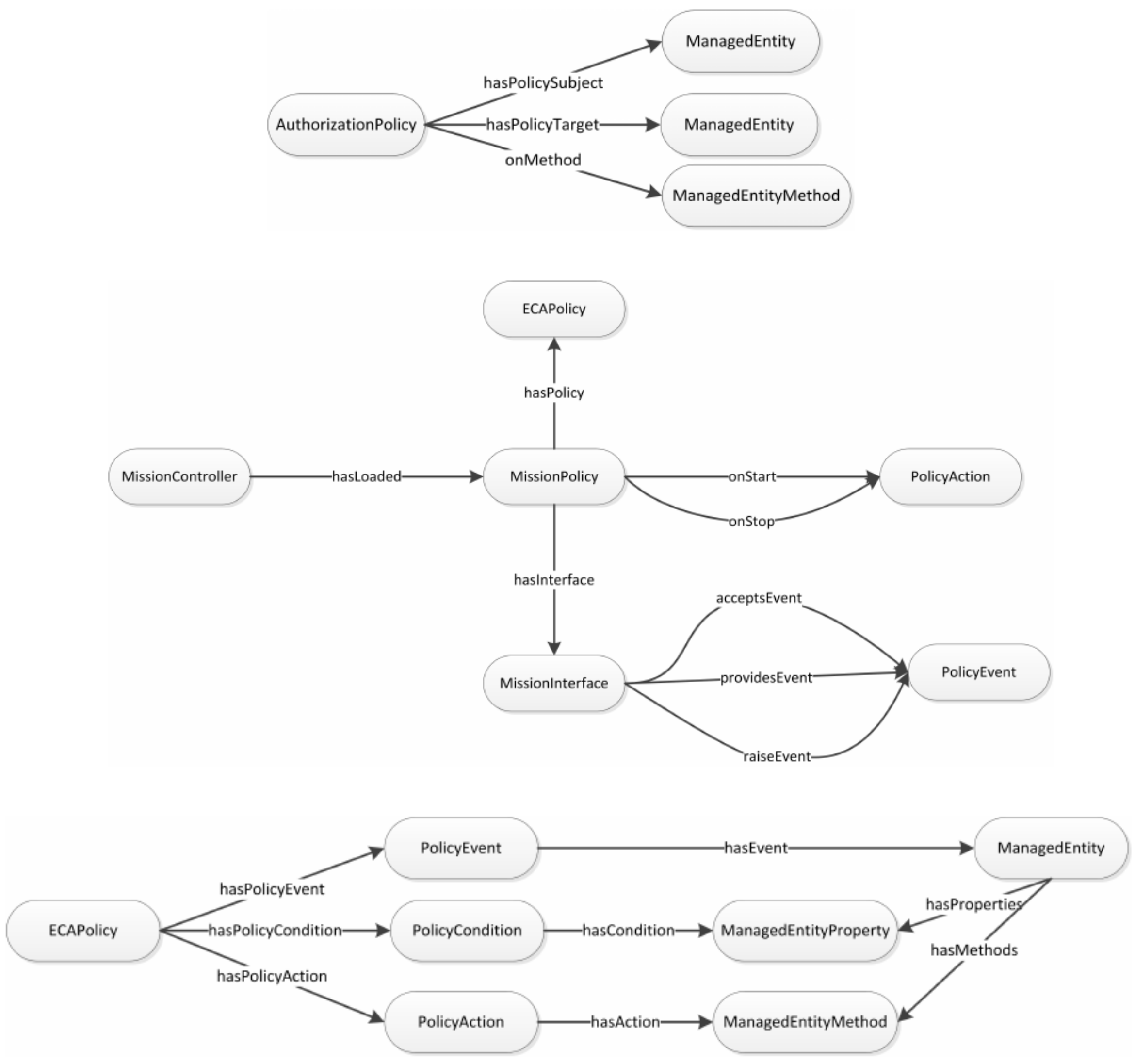


Figure 49. NOVI policy ontology types (Source [NOVI2015]).

Policy Ontology

NOVI's Policy Ontology, as depicted in Figure 49, includes the classic authorization policies and ECA policies. In addition to them, NOVI models an interesting entity from the NECOS' point of view, the Mission Policies, used to define inter-platform duties, *i.e.*, the management obligations that a platform must fulfil against its peer platform in a NOVI federation.

7.2.2 ETSI NFV MANO

The European Telecommunications Standards Institute (ETSI) has defined a framework for Network Functions Virtualization (NFV) and Management and Orchestration Architectures (MANO) [MANO]. More specifically, MANO, as a top-down approach, consists of three main software layers:

- NFV Orchestrator (NFV-O) is responsible for network service management, such as to create virtual function instances to meet service requirements. NFV-O responsibilities include the onboarding of new Network Service (NS) and the NS lifecycle management. Other functionalities include the global resource management (topology of the connected VNFs and PNFs), the authorization of NFV infrastructure resource requests and the policy management for NS instances.
- VNF Manager (VNF-M) manages the lifecycle of the components and services. The VNF-M supervise the management of the VNF instances, *i.e.*, VNF starts from VNF-M descriptor and is managed by VNF-M, also VNF-M determines the health of the VNF.
- Virtualized Infrastructure Manager (VIM) manages NFV infrastructure resources in a single domain. VIM controls and manages the NFV infrastructure resources in one operator's infrastructure sub-domain. Moreover, the VIM is responsible to collect and forward the network performance measurements.

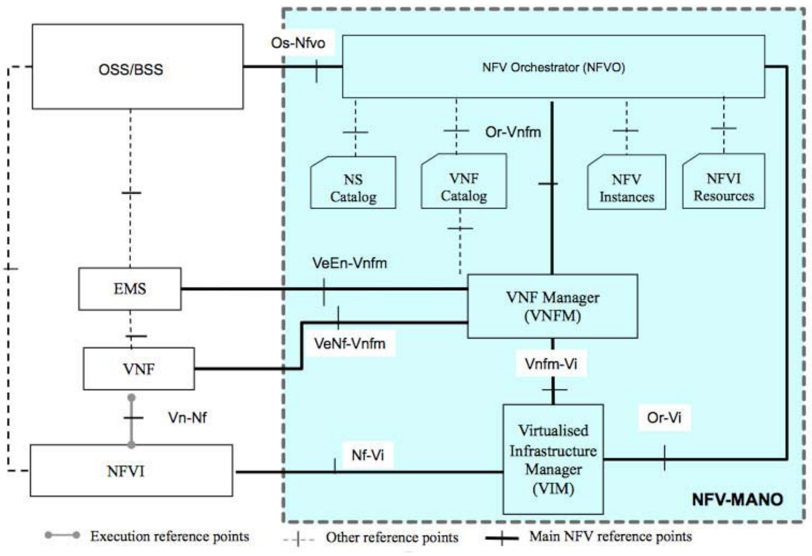


Figure 50. NFV MANO reference architecture (Source: ETSI NFV MANO WI document).

As shown in Figure 50, apart from the aforementioned 3-layers, the NFV MANO consists of 4 types of data repositories (databases that keep different types of information):

- The NS catalogue, which is a set of pre-defined templates that define how services may be created and deployed; the same repository stores the connectivity parameters through virtual links for future use.

- The VNF catalogue, which is a set of templates that describe the deployment and operational characteristics of available VNFs.

- The NFVI resources repository, which maintains information about available/allocated NFVI resources.
- NFV instances repository, which maintains information about all function and service instances throughout their lifetime.

MANO Information Model

ETSI OSM is delivering an open source Management and Orchestration (MANO) stack aligned with ETSI NFV Information Models that focuses on network service orchestration. Information in a network service (NS) is structured into information elements, which might contain a single value or additional information elements that form a tree structure. Information elements are classified as one of the following types: leaf element (single information element), reference element (information element that contains a reference to another information element) and sub-element (information element that specifies another level in the tree). The information elements can be used in two different contexts: as descriptors or as run-time instance records. A descriptor is defined as a configuration template that defines the main properties of managed objects in a network.

The network service descriptor (NSD) is the top-level construct used for designing the service chains, referencing all other descriptors that describe components that are part of that network service. The NSD consists of static information elements that describe deployment flavours of the network service. The NSD is used by the NFV orchestrator to instantiate a network service.

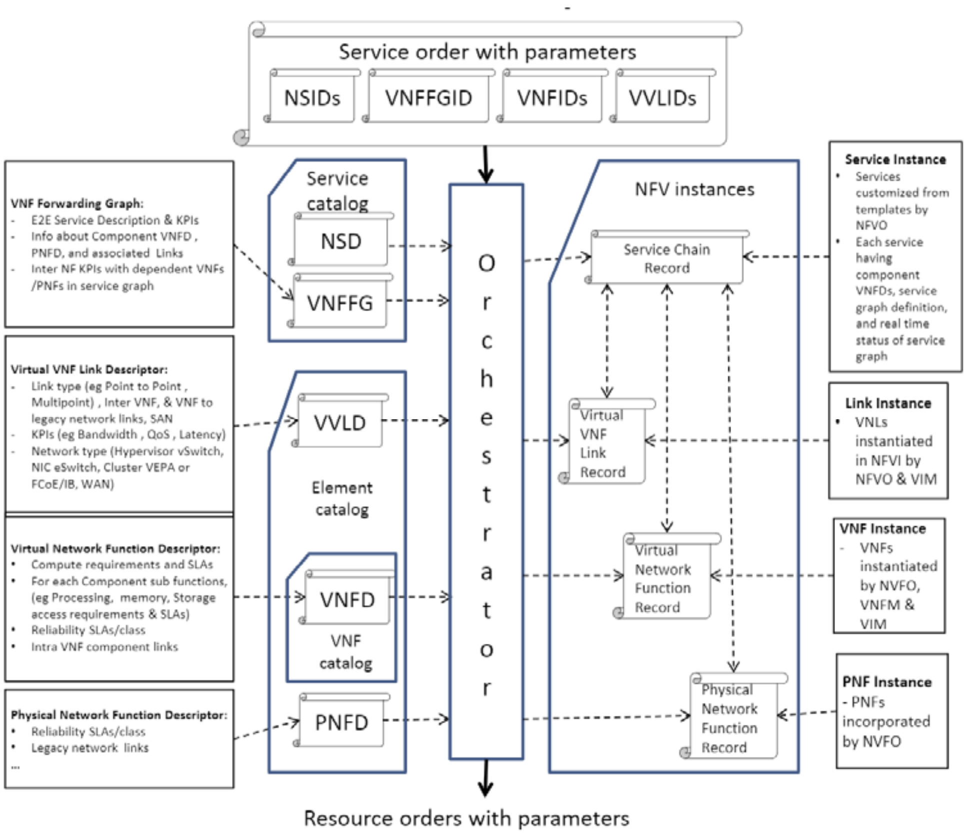

Service order with parameters
NSIDs
VNFFGID
VNFIDs
VVLIDs
VNF Forwarding Graph:
- E2E Service Description & KPIs
- Info about Component VNFD , PNFD, and associated Links
- Inter NF KPIs with dependent VNFs /PNFs in service graph
Virtual VNF Link Descriptor:
- Link type (eg Point to Point , Multipoint) , Inter VNF, & VNF to legacy network links, SAN
- KPIs (eg Bandwidth , QoS , Latency)
- Network type (Hypervisor vSwitch, NIC eSwitch, Cluster VEPA or FCoE/IB, WAN)
Virtual Network Function Descriptor:
• Compute requirements and SLAs
• For each Component sub functions, (eg Processing, memory, Storage access requirements & SLAs)
• Reliability SLAs/class
• Intra VNF component links
Physical Network Function Descriptor:
• Reliability SLAs/class
• Legacy network links
...
Service catalog
NSD
VNFFG
VVLD
Element catalog
VNFD
VNF catalog
PNFD
Orchestrator
NFV instances
Service Chain Record
Virtual VNF Link Record
Virtual Network Function Record
Physical Network Function Record
Service Instance
• Services customized from templates by NFVO
• Each service having component VNFDs, service graph definition, and real time status of service graph
Link Instance
• VNLs instantiated in NFVI by NFVO & VIM
VNF Instance
- VNFs instantiated by NVFO, VNFM & VIM
PNF Instance
- PNFs incorporated by NVFO
Resource orders with parameters

Figure 51. MANO information model (Source: ETSI NFV MANO WI document).

The following four information descriptors are defined apart from the top-level network service (Figure 51):

- Virtual network function (VNF) information element, which is a deployment template that describes the attributes of a single VNF.

- Physical network function (PNF) information element, which describes a physical (legacy) network function and includes only the interconnections (connection points and virtual links). The PNF descriptor is needed if the network service includes a physical device to support network evolution.

- Virtual Link (VL) information element, which describes the resource requirements needed for a link between VNFs, PNFs and end-points of the network service, which could be met by various link options that are available in the NFVI.

- VNF forwarding graph (VNFFG) information element, which is a graph, specified by a network service provider, of bi-directional logical links that connect network function nodes, where at least one node is a VNF through which network traffic is directed.

Software that provides VNFs can be structured into software components, the implementation view of a software architecture. These components can then be packaged into one or more images, the deployment view of a software architecture. These software components are called Virtual Network Function Components (VNFCs). VNFs are implemented with one or more VNFCs, where each VNFC instance generally maps 1:1 to a VM image or a container, as defined in the VDU.

Reference Points

MANO has multiple reference points that appear as interconnection points between the functional blocks, as shown in Figure 50, *i.e., Or-Vi*, *NF-Vi*, *Or-Vnfm*. Designed with open, standards-based APIs, such as NETCONF and REST, and common information models, such as YANG, the *Os-ma-nfvo* interface is exposed through open, standards-based interfaces, such as REST. This design enables upper-level orchestrators, such as Business Process Orchestrators or Service Orchestrators, to automate the entire service bring-up process.

7.2.3 4WARD

The 4WARD project [4WARD] designed an architecture for the provisioning and management of service-tailored virtual networks across multiple administrative domains. 4WARD relies on a centralized coordinator, namely Virtual Network Provider (VNP), for virtual network deployment. In particular, the VNP receives virtual network topology requests from a client, and subsequently partitions the request across the participating infrastructure providers. Then, each infrastructure provider receives his corresponding virtual network segment, which he embeds onto his physical topology. Finally, the topology segments are stitched together to form a virtual network requested by the clients.

In this context, the project has specified an information model for the description of the infrastructure and the virtual network resources. The information model is used by all actors to specify and exchange virtual/physical resource information during resource advertisement, assignment, monitoring, and allocation of virtual networks. The model uses the abstraction “Network Element” to describe nodes, interfaces, links, and paths. Each resource element has a unique identifier and a set of attributes. The information model ensures the binding of different elements, such as the binding of links with paths, interfaces with nodes, and so forth.

Figure 53 shows a UML diagram that expresses the relationships between virtual resources. UML is a modelling language that separates the conception phase from the implementation phase and provides a generic description that is implementation-independent. 4WARD has particularly selected XML for the implementation of the information model, represented with UML.

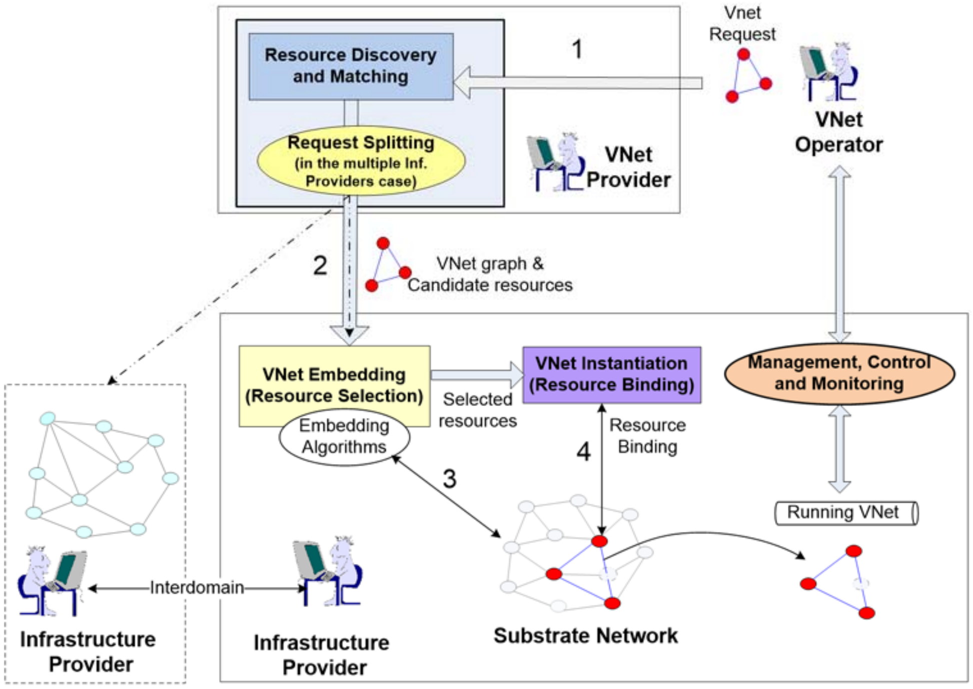

Vnet Request
1
Resource Discovery and Matching
Request Splitting (in the multiple Inf. Providers case)
VNet Provider
VNet Operator
2
VNet graph & Candidate resources
VNet Embedding (Resource Selection)
Embedding Algorithms
Selected resources
VNet Instantiation (Resource Binding)
Resource Binding
4
3
Management, Control and Monitoring
Running VNet
Interdomain
Infrastructure Provider
Infrastructure Provider
Substrate Network

Figure 52. The 4WARD approach to network virtualization (Source: 4WARD project presentation).

The 4WARD information model provides a detailed description of infrastructure and virtual resources to meet the requirements of virtual network provisioning. The different levels of abstractions offered by the model comprise an additional benefit. However, in the context of network slicing, the specific model exhibits considerable limitations. First, there are not sufficient elements and attributes in the model to express service elements (*e.g.*, vNFs) as well as vNF graphs. Hence, this model cannot support the slice specification requirements in all NECOS modes (*e.g., Mode 3* which is associated with service-oriented slice requests). Furthermore, the 4WARD information model does not include descriptors for dedicated infrastructure elements, such as switches, routers, Wi-Fi access points, and base stations. Another limitation of the model is the lack of support for extended platform awareness (EPA), *i.e.*, the exposure of certain capabilities of the infrastructure to the tenant for slice deployment. EPA is a significant feature which needs to be incorporated into information models, given the diversity of features available in modern commodity servers and cloud platforms.

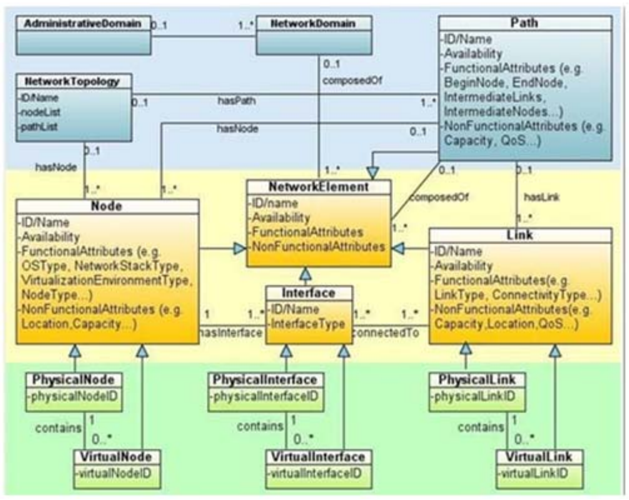


Figure 53. UML diagram of 4WARD information model (Source: [MEDHIOUB2011).

7.2.4 COMS

The Common Operation and Management of network Slicing (COMS) [ietfcoms2018a] aims at providing a comprehensive approach for the overall operation and management of network slicing, for both network slice operators and network slice tenants. Working on the top network orchestrator inside Transport Network region which directly communicates with the network slice provider, COMS enables technology-independent network slice management [ietfcoms2018b]. In this context, COMS provides a technology-independent information model for transport network slicing [ietfcoms2018c].

The COMS idea of a general information model serves the need to fill the gap between technology-agnostic network slicing service requirements, usually desired by the tenants, and technology-specific slices' implementation, typically supported by the service providers. Such a model describes the entities that a network slice consists of, along with their properties, attributes, operations and the way they relate to each other, whilst it remains independent of any specific repository, software, protocol or platform.

The COMS information model uses the data model for network topologies as a base [ietfdata2018] and enhances it with new slice-specific attributes under the "netslice" namespace. COMS uses the YANG data modelling language [RFC7950] to make a technology-independent representation of the transport network slice data model. This information model includes, among others, the following elements, which are represented in Figure 54:

- **connectivity resources**: refer to nodes and links that represent virtual nodes and links exposed to the slice user. The COMS augments these two elements with further new attributes, compared to the model [ietfdata2018], in order to represent requirements, configuration and statistics associated with a node (*i.e.*, sent/received-packets) and QoS information associated with a link (*i.e.*, link-bandwidth-agreement).

- **storage resources**: the location attribute describes the location of the storage unit, and other interesting for NECOS' modelling attributes include access-mode (public or dedicated) and read-write-mode with read-only and read/write options.

- **compute resources**: the location attribute describes the location of the compute unit, and other interesting for NECOS' modelling attribute is access-mode (shared or dedicated).

- **service instance based on predefined function blocks**: some general features can be grouped into function blocks in advance, such as load-balancer, firewall.

- **network slice level attributes**: defines a set of attributes directly applicable to a network slice, such as service-time-start/end and lifecycle-status (*i.e.*, construction, modification, activation, deletion).

COMS is a simple model that allows for extension and covers some of NECOS' modelling needs regarding its networks domain. It is interesting to note that the model defines a set of operations that must be supported for the complete network slice. However, apart from the network slice as a whole, each element insides a network slice should also be able to be operated individually. Operations defined by the COMS are:

- **construct**: construct a network slice,
- **delete**: delete a network slice,
- **modify**: modify a constructed network slice,
- **set_element_value**: set the value of an indicated element in a network slice,
- **get_element_value**: get the value of an indicated element in a network slice,
- **monitor**: monitor the status of a network slice, and
- **enable_report**: enable the active report to the subscribes/management system when the monitored status changes beyond expectation.

The aforementioned operations map to NECOS slicing model but they must be extended to accommodate its different slicing modes.

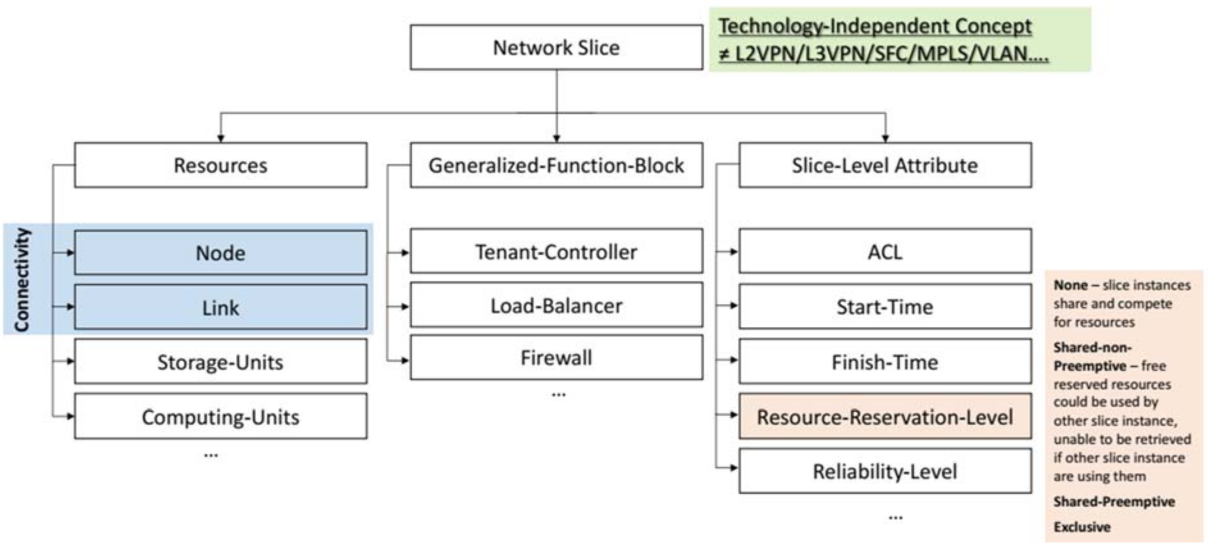

Network Slice
Technology-Independent Concept ≠ L2VPN/L3VPN/SFC/MPLS/VLAN....
Resources
Generalized-Function-Block
Slice-Level Attribute
Connectivity
Node
Link
Storage-Units
Computing-Units
...
Tenant-Controller
Load-Balancer
Firewall
...
ACL
Start-Time
Finish-Time
Resource-Reservation-Level
Reliability-Level
...
None – slice instances share and compete for resources
Shared-non-Preemptive – free reserved resources could be used by other slice instance, unable to be retrieved if other slice instance are using them
Shared-Preemptive
Exclusive

Figure 54. COMS tree of attributes for a slice (Source: [ietfcoms2018b]).

8 References

[3GPP] 3GPP, http://www.3gpp.org/

[3GPP-KPIs] 3GPP, “5G end to end Key Performance Indicators (KPI)”, Release 15, 3GPP TS 28.554, ver. 15.3.0, June 2019.

[5GEX] 5GEx Deliverable 2.2, “5GEx Final System Requirements and Architecture”, December 2017, https://drive.google.com/open?id=1O8KjGolPEAWlit0wRJ_pLykvvanJ2Rk5

[5G-CDN] P. Valsamas, I. Sakellariou, S. Petridou and L. Mamatas, “A Multi-domain Experimentation Environment for 5G Media Verticals”, IEEE International Conference on Computer Communications, Paris, France, 2019.

[5G-MEDIA] S. Rizou, P. Athanasoulis, P. Andriani et al., “A service platform architecture enabling programmable edge-to-cloud virtualization for the 5g media industry”, IEEE Int. Symposium on Broadband Multimedia Systems and Broadcasting (BMSB), June 2018.

[5GINFIRE] A. Gavras, S. Denazis, C. Tranoris et al., “Requirements and design of 5G experimental environments for vertical industry innovations”, IEEE Global Wireless Summit (GWS), Oct. 2017.

[5GNORMA2017] K. Katsalis, N. Nikaein, E. Schiller, A. Ksentini, and T. Braun, “Network slices toward 5G communications: Slicing the LTE network”, IEEE Communications Magazine, Vol. 55 No. 8, 2017.

[5GNORMAd5.1] Z. Yousaf, “D5.1 Definition of connectivity and QoE/QoS management mechanisms intermediate report”, 5GNOMA Project Deliverable, September 2016, http://www.it.uc3m.es/wnl/5gnorma/deliverables/

[5GNORMAd7.2] M. Breitbach, “D7.2: Communication and dissemination final report”, 5GNORMA Project Deliverable, 2017, http://www.it.uc3m.es/wnl/5gnorma/deliverables/

[5GTANGO] P. Twamley, M. Muller, P.-B. Bok, G. K. Xilouris, C. Sakkas, M. A. Kourtis, M. Peuster, S. Schneider, P. Stavrianos, and D. Kyriazis, “5GTANGO: An approach for testing NFV deployments”, European Conference on Networks and Communications (EuCNC) 2018.

[5GTANGO-1] E. Kapassa, M. Touloupou, A. Mavrogiorgou, and D. Kyriazis, “5G & SLAs: Automated proposition and management of agreements towards QoS enforcement”, 21st IEEE Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN) 2018.

[5GTANGO-2] M. Zhao, F. Le Gall, P. Cousin, R. Vilalta, R. Munoz, S. Castro, M. Peuster, S. Schneider, M. Siapera, E. Kapassa et al., “Verification and validation framework for 5G network services and apps”, IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN) 2017.

[5GTANGO-3] E. Kapassa, M. Touloupou, and D. Kyriazis, “SLAs in 5G: A complete framework facilitating VNF-and NS-tailored SLAs management”, 32nd International Conference on Advanced Information Networking and Applications Workshops (WAINA) 2018.

[ALLIANCE20155g] NGMN Alliance, “5G White paper”, Feb. 2015.

[ALTAMANN2008] J. Altmann, C. Courcoubetis, G. D. Stamoulis, M. Dramitinos, T. Rayna, M. Risch, C. Bannink, "GridEcon: A Market Place for Computing Resources", Grid Economics and Business Models, 2008, Springer Berlin Heidelberg, Berlin, Heidelberg.

[BARANWAL15] Gaurav Baranwal, Deo Prakash Vidyarthi, “A fair multi-attribute combinatorial double auction model for resource allocation in cloud computing”, Journal of Systems and Software, 108: 60-76, 2015.

[BBF] BBF SD-406: End-to-End Network Slicing, https://www.broadband-forum.org/5g

[BOZAKOV] Z. Bozakov and P. Papadimitriou, Towards a Scalable Software-Defined Network Virtualization Platform, IEEE/IFIP SDNMO 2014.

[CONTRERAS18] L. M. Contreras, "Slicing challenges for operators", Book chapter in Emerging Automation Techniques for the Future Internet, M. Boucadair and C. Jacquenet (Eds.), IGI Global, 2018.

[DIETRICH2013] D. Dietrich, A. Rizk, P. Papadimitriou, "Multi-domain virtual network embedding with limited information disclosure", IFIP Networking Conference, May 2013.

[DIETRICH2017] D. Dietrich, A. Abujoda, A. Rizk, and P. Papadimitriou, Multi-Provider Service Chain Embedding with Nestor, IEEE Transactions on Network and Service Management, Vol. 14, No. 1, March 2017

[ELHABBASH2019] A. Elhabbash, Y. Elkhatib, G. S.. Blair, Y. Lin, and A. Barker, "A Framework for SLO-driven Cloud Specification and Brokerage", 6th IEEE Workshop on CrossCloud Infrastructures & Platforms, 2019.

[FENDE] L. Bondan, M.F. Franco, L. Marcuzzo, G. Venancio, R.L. Santos, R.J. Pfitscher, E.J. Scheid, B. Stiller, F. De Turck, E. P. Duarte, and A.E. Schaeffer-Filho, "FENDE: Marketplace-based distribution, execution, and life cycle management of VNFs", IEEE Communications Magazine, Vol. 57, No. 1, 2019.

[FUTEBOL] P. Marques, C. Silva, V. Frascolla et al., "Experiments overview of the EU-Brazil FUTEBOL project", 26th European Conference on Networks and Communications (EuCNC), June 2017.

[GALIS2017] "Network slicing terms and systems" slides. Available online: https://datatracker.ietf.org/meeting/99/materials/slides-99-netslicing-alex-galis-netslicing-terms-and-systems-02, Access in 30 May, 2018.

[GENI] M. Berman, J. S. Chase, L. Landweber et al., "GENI: A federated testbed for innovative network experiments", Computer Networks, Elsevier, vol. 61, Mar. 2014.

[GOTO18] Y. Goto, Standardization of Automation Technology for Network Slice Management by ETSI Zero Touch Network and Service Management Industry Specification Group (ZSM ISG), NTT Technical Review, Vol. 16, No. 9, Sept. 2018.

[ietfdata2018] A. Clemm, et al., "A Data Model for Network Topologies", Internet Draft, expires on June 2018. Available online: https://tools.ietf.org/html/draft-ietf-i2rs-yang-network-topo-20

[ietfcoms2018a] L. Qiang, et al. "Technology Independent Information Model for Network Slicing", Internet Draft, expires on July 30, 2018. Available online: https://tools.ietf.org/html/draft-qiang-coms-netslicing-information-model-02.

[ietfcoms2018b] C. Qiang, "COMS Architectural Design Enablers & Artefacts-I, COMS Technology Independent Information Model" Slides. Available online: https://datatracker.ietf.org/meeting/101/materials/slides-101-coms-coms-architectural-design-enablers-artefacts-1-coms-technology-independent-information-model-cristina-qiang-00

[ITU-T Y.3011] Recommendation ITU-T Y.3011 (01/2012), SERIES Y: GLOBAL INFORMATION INFRASTRUCTURE, INTERNET PROTOCOL ASPECTS AND NEXT-GENERATION NETWORKS - Next Generation Networks – Future networks - Framework of network virtualization for future networks.

[ITU-T Y.3111] Recommendation ITU-T Y.3111 (09/2017), SERIES Y: GLOBAL INFORMATION INFRASTRUCTURE, INTERNET PROTOCOL ASPECTS, NEXT-GENERATION NETWORKS, INTERNET OF THINGS AND SMART CITIES - Future networks -IMT-2020 network management and orchestration framework.

[KATSAROS2019] K. Katsaros, V. Glykantzis, P. Papadimitriou, G. Papathanail, D. Dechouniotis, and S. Papavassiliou, MESON: Facilitating Cross-Slice Communications for Enhanced Service Delivery at the Edge, EuCNC 2019.

[KONDO2009] D. Kondo et al., "Cost-benefit analysis of cloud computing versus desktop grids", IEEE International Parallel and Distributed Processing Symposium (IPDPS), 2009.

[KUMAR2017] D. Kumar, G. Baranwal, Z. Raza, and D. P. Vidyarthi, “A Systematic Study of Double Auction Mechanisms in Cloud Computing”, Journal of Systems and Software, Vol. 125, Mar 2017.

[KUMAR2018] D. Kumar, G. Baranwal, Z. Raza, and D. P. Vidyarthi, “A truthful combinatorial double auction-based marketplace mechanism for cloud computing”, Journal of Systems and Software, Vol. 140, 2018.

[LIBERATI] F. Liberati, A. Giuseppi, A. Pietrabissa, V. Suraci, A. Di Giorgio, M. Trubian, D. Dietrich, P. Papadimitriou, and F. Delli Priscoli, Stochastic and Exact Methods for Service Mapping in Virtualized Network Infrastructures, ACM International Journal of Network Management, Vol. 27, No. 6, Nov./Dec. 2017.

[MAAROUF2014] A. Maarouf, A. Marzouk and A. Haqiq, “A review of SLA specification languages in the cloud computing”, 10th International Conference on Intelligent Systems: Theories and Applications (SITA), 2015.

[MEC] ETSI MEC, Multi-access Edge Computing (MEC), MEC support for network slicing, GR MEC 024 v 2.0.5, July 2018.

[MEDHIOUB2011] H. Medhioub, I. Houidi, W. Louati, and D. Zeghlache, Design, implementation and evaluation of virtual resource description and clustering framework, IEEE International Conference on Advanced Information Networking and Applications (AINA), 2011.

[MEDIUM2019] Descartes Labs, "Thunder from the Cloud: 40,000 Cores Running in Concert on AWS", available at: https://medium.com/descarteslabs-team/thunder-from-the-cloud-40-000-cores-running-in-concert-on-aws-bf1610679978.

[MANO] ETSI NFV MANO, https://www.etsi.org/technologies/nfv/open-source-mano

[MESON] MESON: Optimized Edge Slice Orchestration, http://meson-project.gr/

[NFV] ETSI Network Operator Perspectives on NFV priorities for 5G, https://portal.etsi.org/NFV/NFV_White_Paper_5G.pdf.

[NGMN] NGMN White Paper description of network for service provider networks, https://www.ngmn.org/fileadmin/user_upload/161010_NGMN_Network_Slicing_framework_v1.0.8.pdf

[NGUYEN2012] T. Nguyen, S. Bimonte, L. D'Orazio, J. Darmont, "Cost models for view materialization in the cloud", Joint EDBT/ICDT Workshops, 2012.

[NITOS-RAN] N. Makris, C. Zarafetas, S. Kechagias et al., “Enabling open access to LTE network components; the NITOS testbed paradigm”, IEEE conf. on Network Softwarization (NetSoft), April 2015.

[NML] Network Markup Language Working Group, Available online: http://www.ogf.org/gf/group_info/view.php?group=nml-wg

[NOVI2015] J. van der Ham, J. Stéger, S. Laki, Y. Kryftis, V. Maglaris, and C. de Laat, “The NOVI information models, Future Generation Computer Systems”, 42(C), 2015, pp. 64–73, http://doi.org/10.1016/j.future.2013.12.017

[ONAP] Open network automation platform, https://www.onap.org/

[ONAPwiki] Developer wiki: Open network automation platform, https://wiki.onap.org/

[ONAPRef] A. Boubendir, F. Guillemin, C. Le Toquin, M.-L. Alberi-Morel, F. Faucheux, S. Kerboeuf, J.-L. Lafragette, and B. Orlandi, “Federation of cross-domain edge resources: a brokering architecture for network slicing”, IEEE Conference on Network Softwarization and Workshops (NetSoft) 2018

[PAGODA2017] I. Afolabi, A. Ksentini, M. Bagaa, T. Taleb, M. Corici,, and A. Nakao, “Towards 5G network slicing over multiple-domains”, IEICE Transactions on Communications, Vol. 100, No. 11, 2017

[PROMETHEUS] Prometheus Monitoring System, https://prometheus.io/

[RAN] X. Foukas, M. K. Marina, and K. Kontovasilis, "Orion: Ran slicing for a flexible and cost-effective multi-service mobile network architecture", 23rd ACM Annual Int. Conf. on Mobile Computing and Networking, October 2017.

[RFC7950] M. Bjorklund, The YANG 1.1 Data Modeling Language. (M. Bjorklund, Ed.). RFC Editor, 2016, https://doi.org/10.17487/RFC7950

[SLICENET] Q. Wang, J. Alcaraz-Calero, MB Weiss, A. Gavras, PM Neves, R. Cale, G. Bernini, G. Carrozzo, N. Ciulli, G. Celozzi, A. Ciriaco et al., "SliceNet: end-to-end cognitive network slicing and slice management framework in virtualised multi-domain, multi-tenant 5G networks", IEEE International Symposium on Broadband Multimedia Systems and Broadcasting (BMSB), June 2018.

[SONATA2017] S. Draxler, H. Karl, M. Peuster, H. R. Kouchaksaraei, M. Bredel, J. Lessmann, T. Soenen, W. Tavernier, S. Mendel-Brin, and G. Xilouris, "SONATA: Service programming and orchestration for virtualized software networks", IEEE International Conference on Communications Workshops 2017.

[SONATAd4.3] J. Bonnet, "D4.3. Service Platform First Operational Release and Documentation", SONATA Project Deliverable, 2017, http://www.sonata-nfv.eu/deliverables/

[TAFSIRI2018] S. A. Tafsiri and S. Yousefi, "Combinatorial double auction-based resource allocation mechanism in cloud computing market", Journal of Systems and Software, volume 137, 2018.

[TNOVA2014] G. Xilouris, E. Trouva, F. Lobillo, J.M. Soares, J. Carapinha, M.J. McGrath, G. Gardikis, P, Paglierani, E. Pallis, L. Zuccaro, and Y. Rebahi, Y., T-NOVA: A marketplace for virtualized network functions, European Conference on Networks and Communications (EuCNC), June 2014.

[TNOVA_TNSM] M.A. Kourtis, M. J. McGrath, G. Gardikis, G. Xilouris, V. Riccobene, P. Papadimitriou, E. Trouva, F. Liberati, M. Trubian, J. Batallé, H. Koumaras, D. Dietrich, A. Ramos, J. Ferrer Riera, J. Bonnet, A. Pietrabissa, A. Ceselli, and A. Petrini, T-NOVA: An open-source MANO stack for NFV infrastructures. IEEE Transactions on Network and Service Management, Vol. 14, No. 3, 2017.

[TOSCA] T. Binz, U. Breitenbucher, O. Kopp, and F. Leymann, "TOSCA: portable automated deployment and management of cloud applications", iAdvanced Web Services. Springer, 2014.

[WAUTERS2014] T. Wauters, B. Vermeulen, W. Vandenberghe et al., "Federation of Internet experimentation facilities: architecture and implementation", European Conference on Networks and Communications (EuCNC) 2014.

Version History

Version	Date	Author	Change record
0.1	01/04/2019	P. Papadimitriou	Creation
0.2	01/07/2019	P. Papadimitriou	First integrated draft
1.0	08/08/2019	P. Papadimitriou	Final version release