unifying cloud
and carrier networks

Deliverable 2.1: Use Cases and Initial Architecture

Dissemination level	PU
Version	1.0
Due date	30.04.2014
Version date	28.08.2014

This project is co-funded
by the European Union

Document information

Authors

Mario Kind (Editor-in-Chief), Fritz-Joachim Westphal, Andreas Roos, Manuel Paul, DT; Pontus Sköldström, ACREO; Balasz Sonkoly, BME; Rebecca Steinert, SICS; Wolfgang John, Catalin Meirosu, Ahmad Rostami, EAB; George Agapiou, OTE; Antonio Manzalini, Vinicio Vercellone, Diego Daino, TI; Róbert Szabo, Dávid Jocha, ETH; David Verbeiren, INTEL; Hagen Woesner, Tobias Jungel, BISDN; Matthias Rost, TUB; Jokin Garay, Jon Matias, Eduardo Jacob, EHU; Wouter Tavernier, IMINDS; Mario Baldi, POLITO

Coordinator

Dr. András Császár

Ericsson Magyarország Kommunikációs Rendszerek Kft. (ETH)

KONYVES KALMAN KORUT 11 B EP

1097 BUDAPEST

HUNGARY

Fax: +36 (1) 437-7467

Email: andras.csaszar@ericsson.com

Project funding

7th Framework Programme

FP7-ICT-2013-11

Collaborative project

Grant Agreement No. 619609

Revision and history chart

Version	Date	Comment
0.1	February 19th 2014	Initial ToC
0.2	March 26th 2014	Revised ToC, time tables, integration of initial contributions and available content
0.3	April 11th 2014	Adding more contributions, preliminary draft version
0.4	May 13th 2014	Adding requirements, architecture principles, overarching and functional architecture
0.5	May 22nd 2014	Revision of state-of-the-art, use cases, feedback all other sections
0.6	June 10th 2014	Revision according to Stockholm meeting
0.7	June 13th 2014	Clarifications on architecture, draft for WP internal review
0.9	June 30th 2014	Draft for project internal review
1.0	August 28th 2014	Final version including reviewer feedback

Table of contents

List of figures

List of tables

Executive Summary

This deliverable provides the initial proposal for the architecture to unifying carrier and cloud networks. Methods of the architecture design follow 1) state of the art review, 2) definition of use-cases, requirements and focus areas, and 3) identification of key design principles.

By the review of the state of the art over 30 important aspects used for supporting the developments towards the objectives of UNIFY in general, for selecting important design principles and finally for defining the UNIFY architecture were identified. Throughout the definition of 11 selected use-case it was found that most of the relevant aspects are covered by three focus areas, namely 1) infrastructure virtualization, 2) flexible service chaining and
3) network service chain invocation (programmability interfaces). These three use-cases groups were defined in details and were also used to identify the set of 46 requirements. Additionally, important technical aspects, like programmability and orchestration, service provider DevOps framework and the Universal Node concept have been separately detailed in two further use cases. The "Elastic Network Function" use case details how scalability for virtualised network functions can be achieved. Three options are under investigation with a "natural" scalable network function, a semi-transparent scalable service chain and the non-transparent scalable service chain. In addition, four integration and management options are identified with scalability managed as 1) input trigger to, 2) directly by, 3) with support from and 4) separately in application specific orchestration logic. The second use case "Secure & Content aware IP VPN" deals with the combination and chaining of distributed physical and virtual network functions controlled by different entities. It provides a vision of future company networks and its relations with other actors in a combined telco and cloud environment.

Process design methodology was applied for the definition of the architecture. Six fundamental processes were identified to describe the behaviour of the UNIFY architecture: Boot-strapping, Programmability, Verification, Observability, Troubleshooting and VNF Development. Boot-strapping is covering all aspects to enable the whole system environment to work including information in database, information exchange between logical elements basic management tasks. Programmability is covering all process aspects of analysis, as well as configuration of services and sub sequentially resources. Monitoring and verification is covering all operational aspects like collection of information, verification, analytics, failure detection and resolution, quality observations. Verification is used to collectively refer to approaches that compare and contrast expected and detected system states, there as observability is used to describe methods that attempt to measure or estimate metrics or parameters and based on them determine particular system states and troubleshooting is used to collectively refer to techniques that correlate and filter information collected from different entities with the purpose to identify a particular erroneous situation. The VNF Development process supports the team developing functionality for a network function to conform to the DevOps principle "Develop and test against production-like systems".

Finally, the proposed architecture is defined in two initial steps: a coarse granular overarching architecture and a functional architecture, while the detailed system architecture will be defined in deliverable D2.2.

The overarching architecture consists of three layers, the above six processes and a set of reference points. The three layers are service, orchestration and infrastructure. The service layer is responsible for providing translation between user demand and technical descriptions. The orchestration layer has responsibilities for Resource Orchestration (analysing service requests and finding most appropriate resources), Controller Adaptation (providing adaptation between resource requests and the dedicated controllers of the specific resource) and Controllers (logical control of specific resources). A Resource Control Function supports contin¬uous feedback on the required resources for applications. Local Resource Managers in the Infrastructure Layer are managing entities on the physical infrastructure. In parallel, each layer is responsible for providing and operating data. Each of the responsibilities is represented in the architecture as functional entity and between them in total six reference points are defined.

The functional architecture combines the layered overarching architecture with the result of the process analysis and details corresponding functional components including databases and interface properties. Each layer has a dedicated component on observability and performance management with the responsibilities for providing reports and analysis as well as trouble-shooting and verification.

Beside the already mentioned aspects, the resulting design defines and frames how 1) service programming, orchestration and optimization, 2) service provider DevOps and 3) a commodity hardware based networking and execution environment can form a unified production environment. It also identifies a narrow-waist reference point at the Resource Orchestration's northbound interface at which recursive virtualization and stacking of the domains is possible. Moreover the Network Function Forwarding Graph (NF-FG) has been identified as a common representation and cornerstone data structure between many of the internal components representing virtual and physical network functions (on different detailed levels), end-points and connections in-between components.

Overall, UNIFY's design can fill the gap between software defined networks (e.g., ONF's SDN architecture) and network function virtualization (ETSI's NFV).

While the other technical work packages already provided first feedback on the definition of the architecture, also the limitations of this first architecture description are acknowledged, which are to be addressed in our next deliverable (D2.2), e.g., detailed interface definitions related to the reference points; policy interactions; dynamic behaviour with control and data plane split; interaction and interfacing with service management and instrumentation, etc. D2.2 will also contain additional details and will reflect first insights from specification and implementation.

1 Introduction

1.1 Project concept

Socio-economic drivers, progress in Information Technologies (IT), tumbling hardware costs and availability of open source software solutions are creating the conditions for a change of paradigm in designing and operating networks and service infrastructures. *Network virtualization* and *Software Defined Networking (SDN)* seem to be key technology enablers in the direction of meeting requirements such as cost reductions (in CAPEX and OPEX) and new business models.

The vision of the UNIFY project considers the entire network as a "unified production environment" integrating data centre and carrier networks capable of overcoming existing limitations in terms of efficiency of operations, flexibility in service deployment and fast provisioning.

One of the major outcomes of the project will be a new reference architecture including the concept of "dynamic fine-granular Service Chaining". This novel concept enables more flexible service creation by leveraging SDN principles and virtualization techniques.

1.2 Relation with other work packages

The work-packages of UNIFY and a schematic workflow of the activities are shown in Figure 1.1. As shown in the workflow, use cases, related requirements and general architectural aspects are first defined (WP2). So it will integrate and steer the activities of the technical work packages. In particular the definition of service orchestration and programmability solution (WP3), a new management framework and advanced capabilities (WP4) and universal node hardware and software architecture (WP5) will be developed and an evaluation of its viability performed. All WP activities will be finally integrated into a prototype under the technical supervision of WP2.

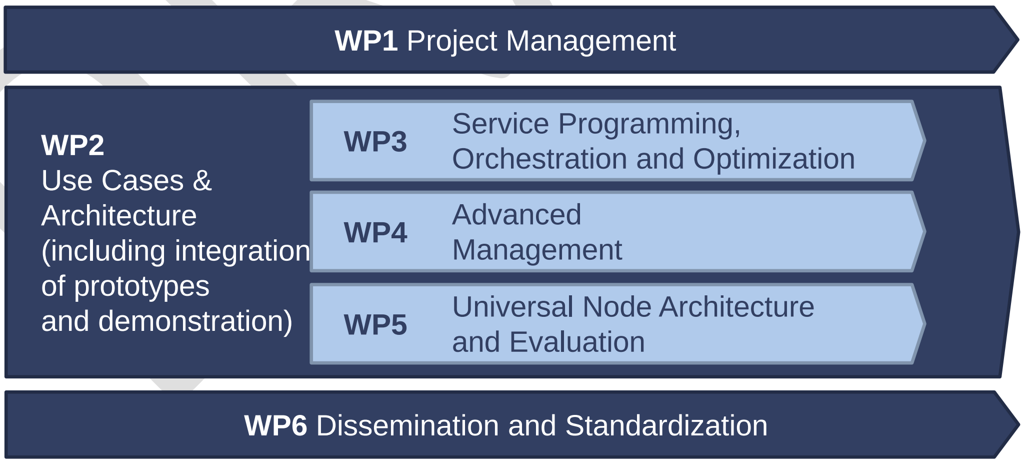

WP1 Project Management
WP2 Use Cases & Architecture (including integration of prototypes and demonstration)
WP3 Service Programming, Orchestration and Optimization
WP4 Advanced Management
WP5 Universal Node Architecture and Evaluation
WP6 Dissemination and Standardization

Figure 1.1: Relation of UNIFY work packages

1.3 Scope of the deliverable

This deliverable represents the initial result of the main objective "*to propose and analyse use case scenarios and to define the UNIFY reference architecture*" that will satisfy all the technical requirements and the business needs for all the different players that are involved in the Information and Communication Technologies. This main objective is split into three fine-granular objectives with:

- Definition of use case scenarios showing how UNIFY can improve today's telecoms processes and operations
- Identification and prioritization of all the design, functional and business requirements
- Definition and design of the different dimensions of a UNIFY reference architecture.

In addition, a first feedback from the technical work packages WP3-5 is already included and the architecture revised accordingly. Overall, the architecture will be finalised with additional feedback from implementation and specifications of the technical work packages in D2.2.

The deliverable is structured as follows: In Section 2 related work is reviewed, in Section 3 use-cases are structured in three use-case groups; in Section 4 the requirements are enumerated, in Section 5 the relevant principles applied in our design are discussed; and in Section 6 and 7 the initial version of the overarching and functional architecture design are presented; finally conclusions and summary of next steps are given in Section 8.

2 State of the art and related work in progress

UNIFY started already with a clear idea about the envisioned architecture. The design of the UNIFY architecture will leverage on already available general principles, aspects and results. And therefore it is useful to extensively analyse the related work and prior-art, which is studied and debated in several standardisation organisations, international projects, initiatives and research projects Recently there is an unparalleled focus and progress on Software Defined Networking and its applicability on the data centre and telecommunication space, so state of the art is a very volatile issue for this topic. There it was decided to summarize the most important aspects based on the work of the different entities in this deliverable and not only focus on the summary of consequences for UNIFY. In order to achieve that, a comprehensive set of organizations and projects were reviewed, their potential influence assessed by the project and briefly summarized in the following sections.

2.1 Software Defined Networking

Everyone knows the IP hourglass (narrow waist) model for the data plane, where packet headers are processed according to local forwarding state for forwarding decision. The IP narrow waist layer contains only four basic functions of logical addressing, forwarding, fragmentation and reassembly. However, there existed no similar abstractions in the control plane to allow computation of the forwarding state. This is targeted by the SDN definition from Scott Shenker [1], the most respected one and foundation of research on SDN lately. The general principles are depicted in the left part of Figure 2.1.

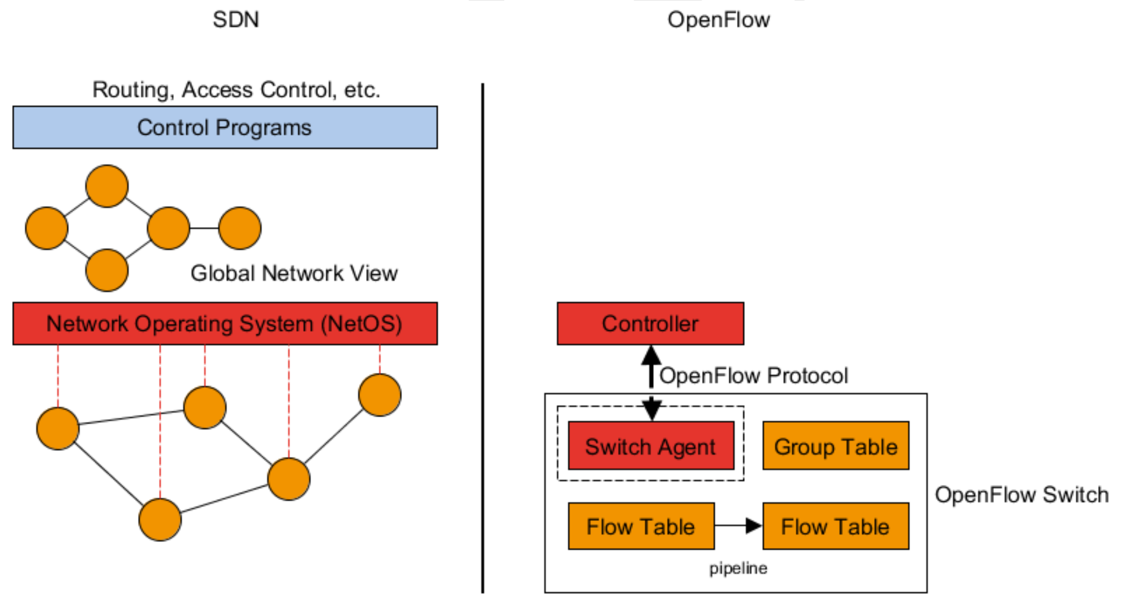


Figure 2.1: SDN based NetOS with Global Network View abstraction and Control Programs and OpenFlow Switch Abstraction and API

This SDN vision [1] seeks for a control plane abstraction

- For a general forwarding model (this includes e.g. simple packet processing like add/remove/modification of headers) compatible with low-level hardware and software;

- Global network state abstraction to be able to compute forwarding decisions;
- Simple configuration of each physical device.

Therefore, the SDN control plane abstraction defines a global network view through the concept of a control plane API, which

- Provides an abstracted network graph and
- Provides configuration control to network elements.

Implementation wise the global network view is maintained in a Network Operating System (NetOS), which runs on servers in the network (may be replicated for reliability and scalability). Information flows from routers/switches to the NetOS to form the global network view and NetOS configures routers/switches to control forwarding. OpenFlow is one example implementation of a node abstraction and API and its relationship to the SDN definition shown in the right part of Figure 2.1.The Global Network View can be used by control programs, which thereby program and define the detailed functions of the network.

To conclude, UNIFY should:

- Reflect the SDN principle of control plane abstraction with different, abstracted levels of details
- Use the concept of control programs (for instanc resource orchestrators, topology database) using different, abstracted levels of information for programming the detailed functions of the network

2.2 Open Networking Foundation

Starting from a particular strong initial focus in enterprise cloud environments and overlay networking, the Open Networking Foundation (ONF) has extended its footprint to all principal types of networks [2]. Several relevant use cases, including particularly cloud networking, L4-L7 support and network service chaining, are being addressed.

From a technical point of view, different working and discussion groups are detailing aspects of a complete SDN framework (detailed list can be found [3]). In summary, the following aspects are targeted:

- General SDN framework and architecture
- Configuration and management principles and techniques
- OpenFlow Protocol extensions with analysis of potential security threats
- Analysis of existing SDN network deployments
- Possibilities to migrate to SDN networks
- Hardware Abstraction Layer installing and configuring on the fly data path elements
- Northbound interface analysis in order to define general requirements
- Wireless and mobile network architecture and network aspects for SDN

- Optical transport networks and technology integration
- L4-7 function chaining
- Carrier grade SDN analysis, definition of requirements and deployment considerations

Most of these aspects and working groups have some relevance to UNIFY. Nevertheless, for sake of brevity, only the basic SDN architecture is described hereafter.

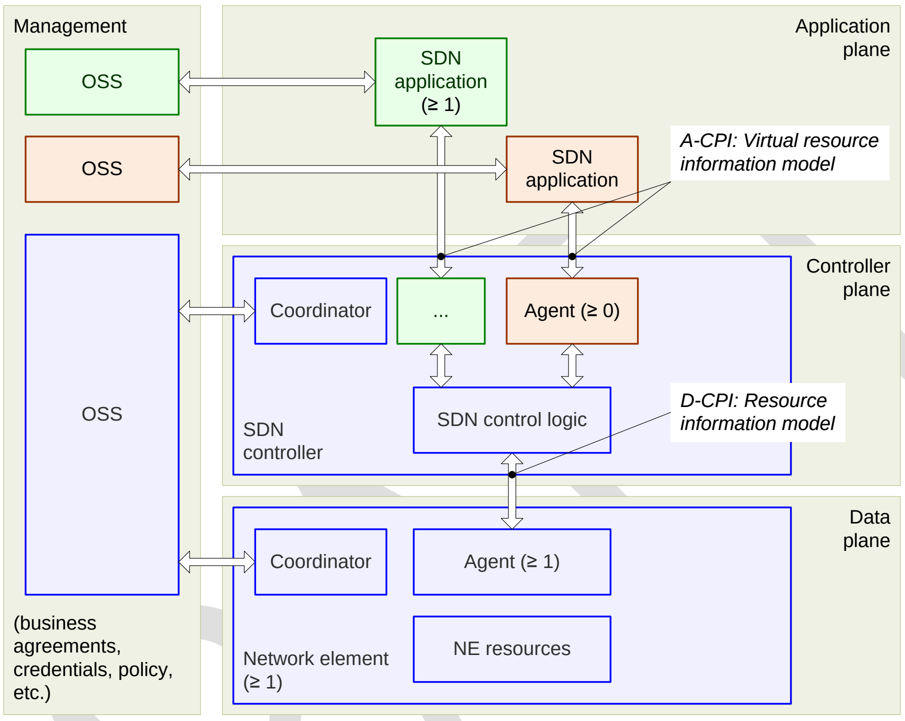


Figure 2.2: ONF SDN overview

As shown in Figure 2.2, the SDN architecture defined by ONF assumes a split into three logical, horizontal planes (application, controller and data) with two interfaces in between (Application-Control Plane and Data-Control Plane Interface (A-CPI/D-CPI)). In addition, some management functionality is assumed in a separate vertical, orthogonal plane. Several fundamental architecture design considerations are made:

- Decoupling of control and data planes
- Logically centralized control
- Hierarchy of control planes for virtualisation, scalability, modularity, security, etc.
- Support of multiple technical or business domains
- Interfaces supporting resource information models
- Abstraction of network resources and exposure of state only

Beside, a number of functional elements are described in more detail, but it should be noted that the ONF model is concentrating on networking resources still.

For UNIFY, all architecture design considerations listed in the previous paragraph should be taken into account. In addition, the various discussed use cases should be taken into account and analysed for further developments.

2.3 ETSI NFV Industrial Study Group

The focus of the ETSI NFV industrial study group [4] is on Network Function Virtualization (NFV) approaches that provide advantages to the networks of operators. Here it is considered "to enable and exploit the dynamic construction and management of network function graphs or sets, and their relationships regarding their associated data, control, management, dependencies and other attributes". Already available is the terminology (see Annex 1), use cases and requirements, which should be considered in UNIFY. The ETSI NFV ISG describes and specifies service models that are intended to represent the roles and interactions of tenants who are involved in the process of providing services which are delivered to the end users via virtualised network functions (VNF). Further, it provides a number of use cases for the virtualisation of network functions that can be part of a service chain, concentrating on virtualisation of network functions.

In addition, ETSI specifies an architectural framework that describes various functional blocks and reference points which are created by the virtualisation of network functions. This architecture as defined in ETSI GS NFV is depicted in Figure 2.3.

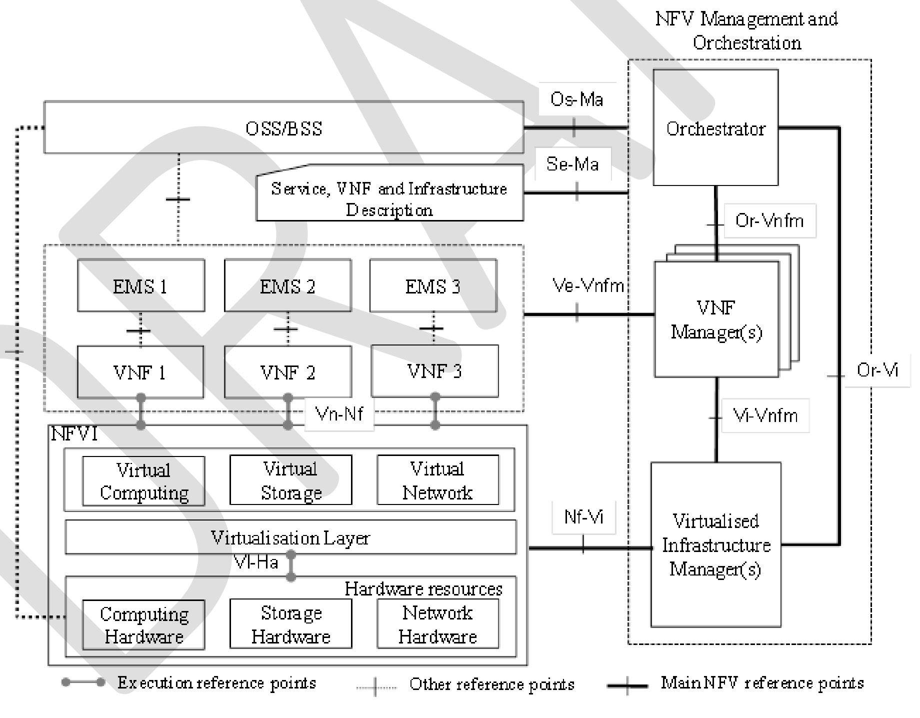


Figure 2.3: NFV reference architectural framework from ETSI

In the above figure, VNF is the virtualization of a network function, the element management system (EMS) is the management functionality of a VNF and the VNF manager is responsible for the VNF lifecycle management. There are two more important elements. The Virtualised Infrastructure Manager is responsible for managing and controlling the interaction of a VNF with the resources under control of the Virtualised Infrastructure

Manager and the orchestrator is responsible for the realisation of services and therefore collection and analysis of information, definition of appropriate actions and management of the NFV infrastructure.

In UNIFY, these aspects should be further analysed:

- Use cases covering virtual network functions
- Requirements targeting virtualisation of network functions
- Architecture with layering, partitioning and virtualisation
- Management elements and their functionality similar to traditional approaches (e.g. specific element manager)

2.4 CloudNFV

The CloudNFV industry initiative [5] identified management as one of the major problems that need solving for ETSI NFV deployments to be successful. CloudNFV defined an architecture that integrates resource orchestration layers with OSS layers in order to manage the (virtual) network functions once successfully deployed. In order to support dynamicity, CloudNFV defined the concepts of Active Contracts and Active Resources. An Active Resource is a representation of a virtual resource that includes policies associated to the resource as well as data related to utilization and health status of the resource. The CloudNFV Orchestration and Management processes perform an analysis of the policies and states associated to Active Resources in order to determine where exactly components of a virtual network function are going to be deployed.

However, a short-coming is that the adaptation of the framework does not really address problems related to fast scalability and elasticity for the case when resources allocated to service graphs are dynamically scaled to address temporarily increases in the demand. The operational aspects rely on existing OSS process and functions for all fault and performance management. Such processes were developed for a rather static network configuration and have difficulties keeping track and analysing virtual infrastructure that is scaled in and out rapidly and migrates often.

In summary, the analysis of the CloudNFV initiative provides the following ideas for further architecture definition in UNIFY:

- Network function virtualisation management along the eTOM process frameworks (bootstrapping and provisioning)
- Template based, abstracted resource description
- Resource orchestration by using a policy framework

2.5 European Research Projects

2.5.1 FP7 MobileCloudNetworking

The MobileCloudNetworking (MCN) project defined [6] a set of large monolithic network functions as services (such as RANaaS, EPCaaS, etc.). Resources are provided through an IaaS framework of atomic services in terms of compute, network and storage, aggregated through a cloud controller with a set of support services provided by the cloud execution environment, effectively providing a PaaS layer. Load Balancing aaS, DNS aaS, Monitoring aaS, AAA aaS, Database aaS are part of this PaaS layer and the project is actively developing them. For higher-level services, such as EPCaaS, the project released blueprints on how they make use of the atomic and the support services provided by the PaaS layer.

There are fundamentally different targets in the general requirements for the MCN and UNIFY architectures. MCN focuses on supporting virtual network functions being monolithically deployed in a combined cloud and network environment. In contrast, the focus on UNIFY is on more fine-granular as well as dynamic service deployment, meaning that large monolithic network functions are split into sets of components that enable reuse and has the potential for a more efficient sharing of resources.

In total, the Mobile Cloud Networking project provides the following ideas:

- Comprehensive use case on mobile networks
- Model of splitting applications in more independent, smaller network functions
- MCN identified and now develops a set of services to be provided by the execution environment supporting the "monolithic aaS" virtualized functions

2.5.2 FP7 ICT Split Architecture (SPARC)

The SPARC project [7] was the first attempt to transfer the idea of OpenFlow and its definition of SDN into a carrier environment. The intention was to break the design lock of monolithic network elements separating forwarding, control and processing blocks. Based on a set of use cases, the SPARC architecture [8] was developed with the key characteristic of hierarchical controllers which are small applications responsible for performing actions on a certain part of the network address space. The OpenFlow protocol was extended to signal all necessary capabilities and match-actions between the different controllers.

In addition, the SPARC project aimed to cover carrier aspects like operation, administration and maintenance (OAM), scalability, virtualization and isolation, resiliency, service creation, control channel bootstrapping and topology discovery, energy efficiency, Quality of Service (QoS) and multilayer analysis.

In total one should take the following aspects into account

- Revisiting use cases and requirements
- Hierarchical controller concept
- Carrier grade network functions

2.6 OpenDaylight

OpenDaylight (ODL) is an Open Source Software project under the auspices of the Linux Foundation. Its goal is furthering the adoption and innovation of SDN through the creation of a common industry-supported platform, with a modular, pluggable, and flexible controller at its core. It is intended for both usage scenarios: network and network services or as network part in the OpenStack (see section 2.8 for details). More details on components of OpenDayLight can be found in Annex 2.

The project has designed an architecture which defines separated layers, several, extensible north- and southbound interfaces to applications and infrastructure as well as a modular set of functions of the controller platform. OpenDaylight supports the Java based OSGi framework and bidirectional REST for the northbound API and for example OpenFlow 1.0, 1.3, OVSDB as south bound interfaces. Another aspect is that the business logic and algorithms reside in either of these (northbound) applications, which use the controller to gather network intelligence, run algorithms to perform analytics, and finally use to orchestrate the new rules.

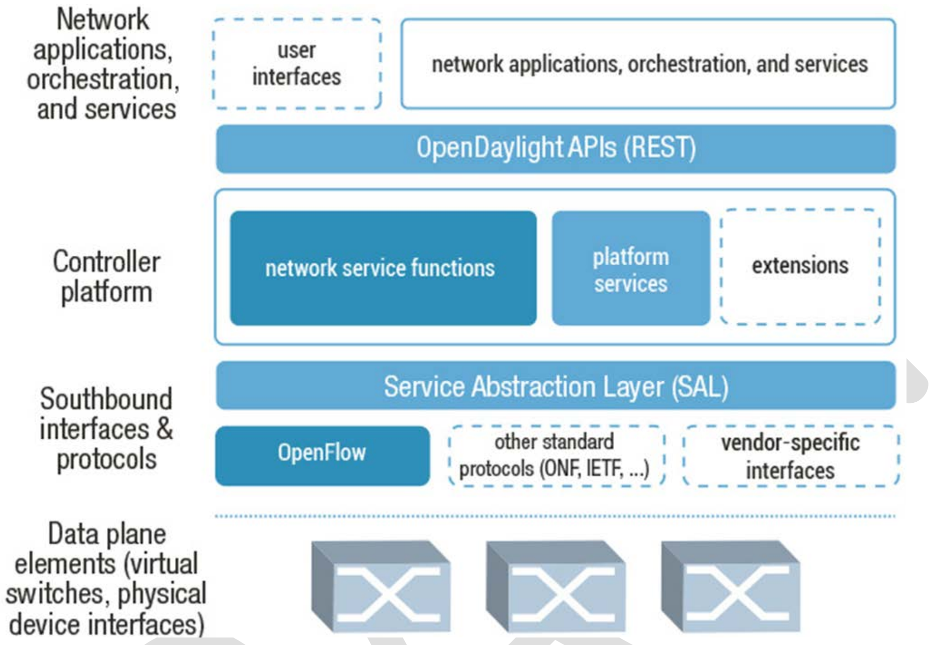

Network applications, orchestration, and services
user interfaces
network applications, orchestration, and services
OpenDaylight APIs (REST)
Controller platform
network service functions
platform services
extensions
Southbound interfaces & protocols
Service Abstraction Layer (SAL)
OpenFlow
other standard protocols (ONF, IETF, ...)
vendor-specific interfaces
Data plane elements (virtual switches, physical device interfaces)

Figure 2.4: OpenDaylight high level view

The controller is implemented strictly in software and is contained within its own Java Virtual Machine (JVM). As such, it can be deployed on any hardware and operating system platform that supports Java.

Open Services Gateway Initiative (OSGi) defines ODL's modular architecture:

- Allows dynamically loading bundles
- Allows registering dependencies and services exported
- Provides means for exchanging information across bundles

OpenDaylight is a multi-project, with a very flexible, modular organisational structure currently running 15 sub-projects. Components from these sub-projects are brought together in a simultaneous release and in "editions" (for example base, virtualization, service provider) in order to provide user-oriented releases.

The following aspects are of importance for UNIFY:

- Split between data and control, layered approach
- User groups targeted by different editions

- Support of multitude of north- and southbound protocols, as well as flexible combination of network functions

2.7 IETF and IRTF

2.7.1 Service Function Chaining (SFC)

The Service Function Chaining (SFC) working group discussions have moved towards "Service Function Graphs" or "Service Function Composition" [9]. The work is currently concentrating on use cases and architecture. A major discussion addresses the granularity of the service chain classification and identification. This particular relation to the definition and integration of service graphs in UNIFY are of high interest and an overview follows in the next paragraphs. More details can be found in Annex 3.

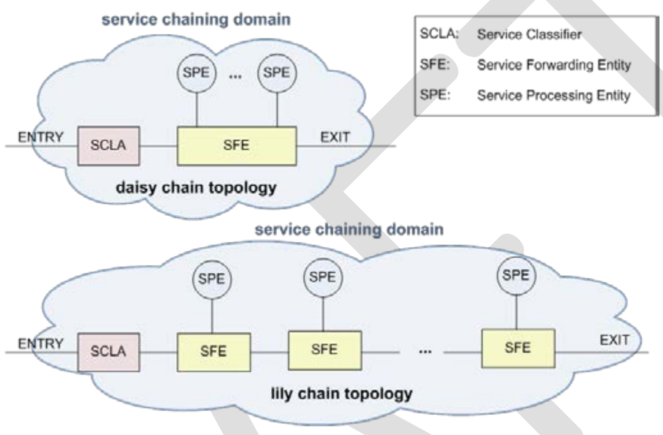


Figure 2.5: Service Function Chaining architecture and components proposed by IETF

The desired target is to overcome today's limitations in terms of dynamicity, flexibility, scalability, optimal usage of resources, etc. basically incurred by a strong coupling to the physical network topology and the capabilities of special purpose, expensive HW elements. In order to overcome these problems a flexible and scalable architecture is defined, see Figure 2.5. It consists of the following components (Figure 2.5):

- Service Classifier (SCLA)
- Service Forwarding Entities (SFEs)
- Service Processing Entities (SPEs)

Incoming packets at the entry point are classified into different service flows by SCLA based on policies or flow characteristics. SFEs forward packets to SPEs according to the configuration of service chains, while SPEs implement the packet processing tasks (network functions). Please note that the architecture is still under discussion and subject to change and other alternative approaches are also under discussion. The latest information can be found in [9].

The combination of the elements could be done in several ways like daisy and lily chain topology or combined hybrid ones (see Figure 2.5). For implementation, a number of additional possibilities must be taken into account like implementation in hardware or software, on one or distributed elements, etc.

Several concepts and ideas should be taken into account:

- Use cases and requirements giving insights in service function chaining
- Layered approach, distinction between processing and forwarding elements
- Control over forwarding elements in order to construct SFPs
- Support conveying metadata between network functions

2.7.2 Network Virtualization Overlay (NVO3)

The NVO3 working group's work on the problem statement for L2/L3 network overlays and NVO3 framework is finished (close to publication [10]). The architecture work covers control plane and management plane concepts. In addition several applicability statements on cloud/VM orchestration, resource mobility, and a number of solution proposals are available and should be taken into account in the design of use cases, requirements and the architecture itself.

2.7.3 Interface to the Routing System (I2RS)

This IETF working group addresses a standardized, programmatic, asynchronous interface for a state transfer into and out of the routing system, in order to improve access to routing-system-level information and mechanisms to support applications in accomplishing application-level goals [11]. The approach does not focus on separating out or offloading forwarding capabilities to an external controller. While suggested use cases are in context to SDN and NFV, I2RS has not yet focused on aligning terminology or mapping interface models (northbound, southbound) to other industry or standards bodies. For this reason it is by itself of interest for UNIFY.

2.7.4 Virtual Network Function resource Pooling (VNFPool)

The discussion list on Virtual Network Function (VNF) Resource Pooling focuses on shared infrastructures with pooled resources, addressing the challenge of how to achieve reliability and high availability in this environment. Currently identification of an appropriate set of problems regarding to VNF reliability and exploring of potential solutions for VNF transition, including service state synchronization, are in focus.

This activity provides relevance to UNIFY in:

- Dynamic allocation between virtual network functions and cloud computing workloads
- Pooling and relocation of VNF resources rapidly, but also reliably

2.7.5 Software Defined Networking Research Group (SDNRG)

IRTF established SDNRG as a research forum open to both industrial and academic contributions [12]. The charter states that it "investigates SDN from various perspectives with the goal of identifying the approaches that can be defined, deployed and used in the near term as well identifying future research challenges", and as such is an attractive venue for presenting research results achieved in UNIFY. Recently, drafts addressing terminology and taxonomy of different SDN approaches have been submitted and should be taken into account by UNIFY.

2.8 OpenStack

OpenStack is a cloud operating system that controls large pools of compute, storage, and networking resources throughout a data centre. Details can be found in Annex 4.

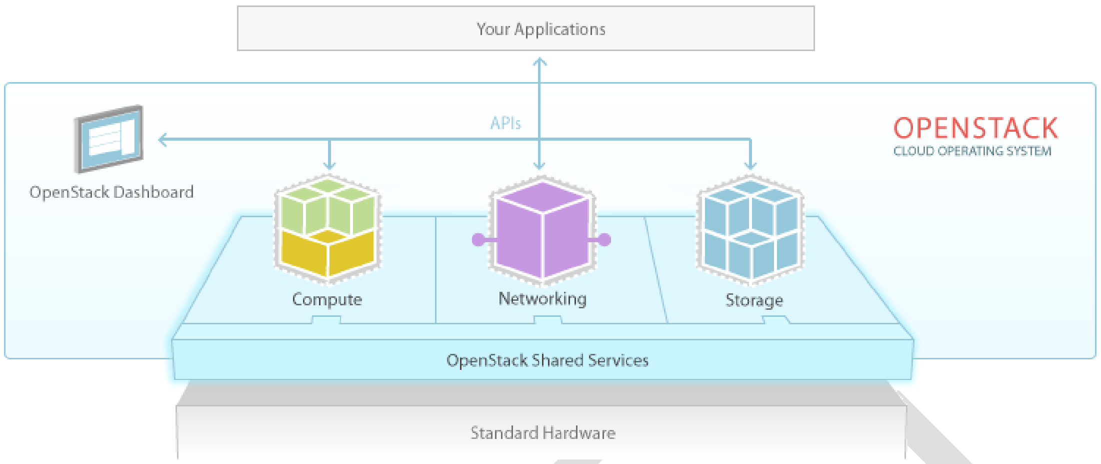


Figure 2.6: OpenStack overview

As shown in Figure 2.6, OpenStack consists of a modular set of components addressing the different resource types and common, shared services with a defined set of interfaces in-between. Each of the components provides an Application Programming Interface (API) to access it, which can be used in the form of a REST (REpresentational State Transfer) API based on HTTP, or as CLI (Command Line Interface) [13]. OpenStack is available as open-source implementation with industry support for adaptations, support for pluggable hypervisors and networking components. The available orchestration component of OpenStack works on a limited set of cloud application parameters and performs a simple optimization on mapping the request to resources/locations taking compute and storage resources into account.

The UNIFY architecture design should consider:

- An architecture clearly defining responsibilities and interfaces
- Following a modular approach, with interchangeable components
- Organisation of processing, storage and network resources
- Descriptions of resources and optimisations

2.9 TMForum/eTOM processes

The TMForum has defined a complete suite covering the management and business aspect of operating networks called Frameworx 13.5 [14]. TMForum's history resides in the telecommunication network area and it lately adopted IT-related aspects as well as latest technology trends. UNIFY specifically targets carrier networks and could borrow basic principles and information, which have proved to be a good basis for understanding and covering operational aspects in the design of carrier architecture.

More specifically, the Business Process Framework eTOM is most relevant in analysis and design phase of the UNIFY project. It provides a hierarchy of typical business processes and requirements covered in an enhanced telecommunications operations map (eTOM). Each hierarchy adds more details to an operational processes starting from very high-level abstract definitions and relations down to very fine-granular focused process descriptions. In addition information models for each process are defined. In practice, one does not have to follow all details and can adapt them to individual needs.

The Business Process Framework consists of three major parts:

- Strategy, Infrastructure & Products which covers more planning and general management activities
- Enterprise management which covers the supporting activities financial, human resources or risk management
- Operations area which defines the core activities of an enterprise and describes all the daily operational aspects

As the primary focus of UNIFY is on the operations area, the other areas have lower relevance for UNIFY as the focus is neither on the operation of an enterprise as such nor on the detailed general and strategic planning issue, which will be most likely required for a real deployment only.

For the upcoming developments the following aspects of TMForum should be taken into account:

- Sequential design processes starting from basic, abstract level and detailing to fine granular levels
- Use of process maps and process descriptions to design and verify architecture
- Select only what is useful, redesign certain aspects or skip unnecessary features

2.10 DevOps

The DevOps concept comes from the IT world, and is a method that focuses on collaboration or even integration of software developers and IT operators with the goal of fast, frequent and high-quality releases. This includes practices and ideas like continuous deployment or continuous delivery very much in line with UNIFY goals of high service velocity.

The current understanding of DevOps can be summarized by four key principles/processes [15]:

- Develop and test against production-like systems
- Deploy with repeatable, reliable processes
- Monitor and validate operational quality
- Amplify feedback loops

While these principles stem originally from the IT DevOps movement (mainly Web2.0 businesses), they can be beneficial in other business domains as well, for example, in the telecommunication service provider settings, as studied in UNIFY.

Many tools are already developed to implement the ideas and visions of DevOps. As SDN defines software based applications as the building blocks for network function the DevOps approach is valid for software defined network architectures like UNIFY.

2.11 BroadBandForum (BBF)

Where the previous standardisation organisations, projects and industry initiatives have been targeting specific components, aspects or protocols, the BBF adds value in the design and operation of the broadband access networks as a whole with all its specific network functions. SIMR (Service Innovation & Market Requirements) is the working group in BBF that drives work directions and requirements in order to integrate innovation, through the evaluation of market and technology trends and the identification and the assessment of the potential enablers and disruptors [16]. Current aspects under investigation are SDN, Cloud Services and Flexible Service Chaining.

Therefore UNIFY should analyse the use cases and requirements works and should take resulting network functions and design principles into account.

2.12 Summary on state of the art

The design of the UNIFY architecture will leverage on already available general principles, aspects and results. The first major input is the analysis of the different standardisation organisations, projects and industry initiatives and practices. This analysis has revealed the following important aspects, which will be considered for the UNIFY project and architecture definition.

There is a list of major, overarching recommendations which should be carefully applied in the UNIFY architecture development:

- Select from available information only what is useful, skip or redesign any unnecessary aspect. Do not restrict the design to cover existing standards or recommendations.
- Analyse existing, described use cases in order to align with existing and on-going discussions and avoid repetition of work (covered in section 3)
- Analyse existing, listed requirements in order to align with existing and on-going discussions and avoid repetition of work (covered in section 4)
- Analyse terminology in order to align with common best practise
- Support of multitude of north- and southbound protocols (will be covered in technical work and integrated prototype)

While having a number of architectures and implementations under investigation, a number of design principles were found, which will be applied (details covered in section 5):

- Sequential architecture development processes from basic, coarse to fine granular detail levels
- Abstraction describing resources and their properties, best in templates
- Use of layering and partitioning, hierarchies

- Recursion used for partitioning of domains, problem space reduction and better reusability of implementations
- Flexible combination of network functions
- Use of process map and process descriptions
- Process framework covering operation of telecommunication networks and extension to integrate DevOps philosophy and its four principles:
  - Develop and test against production-like systems
  - Deploy with repeatable, reliable processes
  - Monitor and validate operational quality
  - Amplify feedback loops

The Architecture will respect the following aspects (application can be found in section 6-8):

- Covering resources types such as processing, storage and network
- Resource orchestration by using a policy framework
- Providing splits / layering of
  - Applications in more independent, smaller network functions
  - Processing and forwarding elements
  - Responsibilities and interfaces
- Modular approach, allowing for exchange of components
- Allow for optimisations
  - Allow dynamic allocation between virtual network functions and cloud computing workloads
- Allow pooling and relocation of VNF resources rapidly
- Allow control over forwarding elements

Important initial aspects of the architecture have been already identified and are summarised in the following. Those aspects are fundamental to explain basic concepts and ideas. Further details will be given in the architecture sections 6 and 7. UNIFY started with the idea to bridge between application services and infrastructure in a more efficient, open and flexible way. Application services and infrastructure could be identified as essential layers, the bridge between them will be the orchestration layer resulting in a three layer architecture. In-between the layers, different information will be exchanged. Here, everything related with application services, the dedicated resources (links, forwarding, computing, etc.) or description of the infrastructure is represented as graph (service or network), more specifically as Network Function Forwarding

Graph (NF-FG), a logical construct. In the following, orchestration will be used as a synonym for all components required for analysing, programming, troubleshooting or exchanging of NF-FG. One process of the orchestrator is programming, where service requests are translated to NF-FG, these NF-FG instantiated, configured and finally set to operational state.

3 Use Cases

One of the first main goals of UNIFY is to identify use cases of interest, which articulate special concerns of cloud and carrier networks interworking, and hence can be used to derive requirements and later help in designing the supporting architecture and finally assess the suitability and comprehensiveness of the design. The sources for the use cases are input of the analysis of the state of the art (see section 2) and operators involved in UNIFY. The use cases described herein are used to exemplify specific functions expected in unified production environment – cloud integration, advanced orchestration and automation mechanisms, programmability and general virtualization. During the exploration of use cases (see detail in Annex 6), three disjoint but complementary focus areas are identified, which are used to summarize our view on the expected functions.

The use cases are grouped in three focus areas based on their main functional/technical characteristics and target.

- Infrastructure Virtualization (Use Case Group 1 (UCG1))
- Flexible Service Chaining (Use Case Group 2 (UCG2))
- Network Service Chain Invocation for Providers (Use Case Group 3 (UCG3))

They group the main concepts from the Functional/Architectural design space depending on main actors/players, the Management/Operational concepts developed in WP3 and WP4 and the functional aspects of the Universal Node that is being designed, implemented and evaluated in WP5. The differentiation between the groups is as follows.

The first use case group (UCG1) presented in Table 3.1 is mainly relative to aspects which deal with virtualisation of nodes or network functions on nodes. In addition these node internal aspects (covering mainly Universal Node concepts) focus on the deployment and operation of virtual network functions, while the second UCG in Table 3.2 deals with the flexible combination and chaining of services represented by network functions. Here orchestration and programmability of an operator's internal domain is important. The third UCG (see Table 3.3) is dealing with all aspects concerning programmability of network service chain Invocation for Providers from two different viewpoints: a more simplified one targeting programmability interfaces within the same legal entity (for instance for organizational reasons) and the more sophisticated one dealing with inter-carrier and -domain viewpoints. Aspects of monitoring, verification and support of virtual network function development are to large extent orthogonal to all UCGs and will be briefly described in each of the sections as well. All three UCGs are detailed hereafter with each one being described separately, highlighting the major aim of the group with network diagrams.

Table 3.1: Use Case Group 1

Use Case Group 1 (Infrastructure Virtualization)	
UC 1	Reuse of existing fixed network operator resources by mobile network operator

UC 2	Interoperability in multi-operator environments - share of network functions across different infrastructure/network domains
UC 3	Multi Access Technology Support
UC 4	Fulfill rapidly evolving and heterogeneous service demand

The first group of use cases deals with the service chain elements in a single domain of heterogeneous infrastructures.

Table 3.2: Use Case Group 2

Use Case Group 2 (Flexible Service Chaining)	
UC 5	Virtual Residential Gateway – Border Network Gateway (BNG)
UC 6	Fast & Flexible provisioning of value added services from operator
UC 7	User controlled NSC and VM functions
UC 8	3D Life-logging

The second group deals with the provisioning and management of service chain functions in an intra-domain scenario.

Table 3.3: Use Case Group 3

Use Case Group 3 (Network Service Chain Invocation for Provider)	
UC 9	Wholesale operator – Network Service Chain as a Service
UC 10	New Service Provider Activation
UC 11	OTT QoS support

The third group of use cases considers service chain functions in an inter-domain scenario and aspects of a network service chain invocation.

Following, the main concepts that dominate each of the UCGs is described, while a more detailed description of all use cases is given in the Annex 6. Given the complexity and manifold aspects representing the unified production environment and which are partly covered in one or the other use case, it was required to concentrate the aspects in a low number of use cases only. Therefore the two most important aspects in UNIFY, the operation of virtual network functions on Universal Nodes and operational and management activities in a unified production environment, have been selected as baseline for two use cases and are enriched with detailed aspects of the eleven use cases. Of course many of the aspects, "condensed" in these two use cases are partly covered by the other use cases as well, but for further studies in the project it is worth to deal with dedicated use cases, focusing on essentials of UNIFY. More details can be found in section 3.4. The two use cases are:

- Elastic Network Function covering a generic Virtual Network Function on a Universal Node and highlighting the flexibility required in a UNIFY environment (see section 3.4.1)
- Secure & Content aware IP VPN covering the operational and management activities of an UNIFY environment and highlighting the intra and inter-domain activities especially for service maintenance (see section 3.4.2) and addressing end-to-end relations

3.1 Use Case Group 1 - Infrastructure Virtualization

The transformation of networking appliances into virtualized network functions with their supporting infrastructure in order to reduce both CapEx and OpEx is strongly envisioned nowadays. Assuming that different technology approaches to virtualise infrastructure are already available, the most important aspect in the context of UNIFY is the deployment of virtual network functions with respect to the different tenants on the infrastructure. Considering the identified use cases (see Annex 6), the infrastructure virtualization must interact with an orchestrating element in order:

- to deploy VNF instances with resource guarantees
- to start/stop/pause/resume/migrate VNF instances
- to change physical resources assigned to VNF instances
- to efficiently and secure share all of the physical resources
- to steer traffic among VNFs (internal)
- to steer traffic to/from the external interfaces
- to monitor resources used by VNFs and traffic steering

The concept of infrastructure virtualization is best illustrated with the following example. Network operators require Broadband Network Gateways (BNG) for customer centric processes like authentication, policing, etc., see use case 5 for more information on the virtualisation of the home gateway. They are located at the edge of the access network located at a point typically referred to as Edge Point of Presence (POP). BNG are physical devices hosting special network services or software fulfilling the processes, sometimes even supported by hardware acceleration. A first step would be that this network function (NF) is virtualised as a virtual network function (VNF), e.g. NF2 to VNF2 as illustrated in Figure 3.1. Another important aspect is the migration of one network function on a dedicated node, like firewall, from the Edge Point of Presence (POP) to the cloud data centre, as seen with VNF4 in Figure 3.1. For instance, in the case of migrating firewall operations in the cloud, access policies will not need to be defined locally in the edge appliance rather will be defined and handled in a centralized and more efficient manner. Moreover, infrastructure virtualization can easily accommodate techniques such as bare-metal restore, i.e., operating system, applications and data restoration to a piece of hardware, even for hardware configurations that are different from the one that the backed-up data were obtained from.

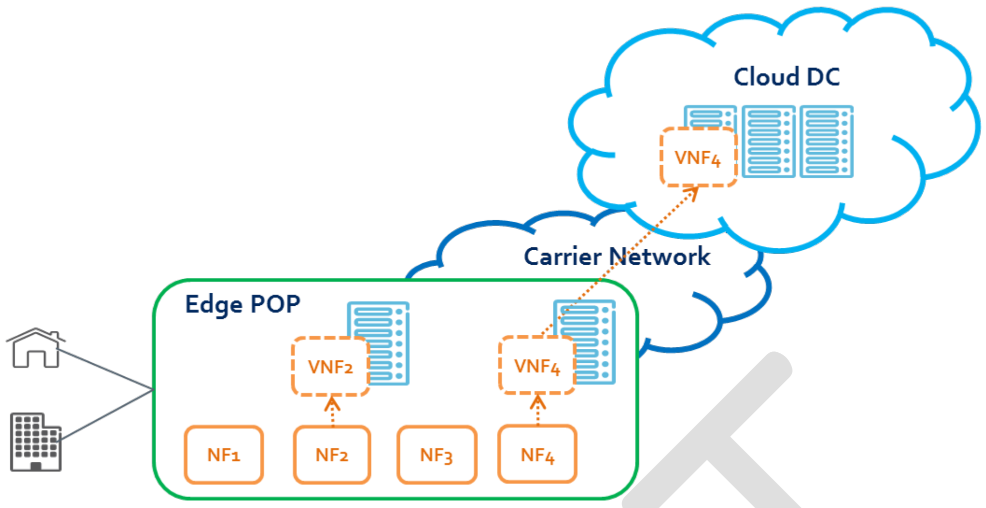


Figure 3.1: Infrastructure virtualization: virtualisation of network appliances and functions

3.2 Use Case Group 2 - Flexible Service Chain

After the first revolutionary step with virtualization of network functions, we envision a second step with the flexible and dynamic composition of network functions within a single administrative domain. We call this concept "Flexible Service Chain". The required functionality is orchestration, which will be integrated as intermediating layer between resources and network services or applications. The important benefits of orchestration are:

- Integration of existing network functions as well as flexible and optimal instantiation of virtual network functions
- Interworking of (abstracted) resources in network and data centre
- Selection of (Abstracted) interfaces for network services and applications enabling programmability[1] of the service graphs
- Instantiation and management of network functions

In Figure 3.2 below, the concept is exemplarily illustrated, while a complete view on the orchestration process is given in section 6.

The orchestration function has the task to find an appropriate realisation of the service chain by instantiating and appropriately connecting network functions. As generalized service chain offerings are envisioned multiple paths between endpoints (see figure below), the general term "service graph" is introduced, which highlights that interconnections can be more complex.

[1] Programmability is the ability of the UNIFY architecture to accept and analysis service requests and to instantiate and configure network functions in the infrastructure. More specific, programmability allows executing code at capable resources.

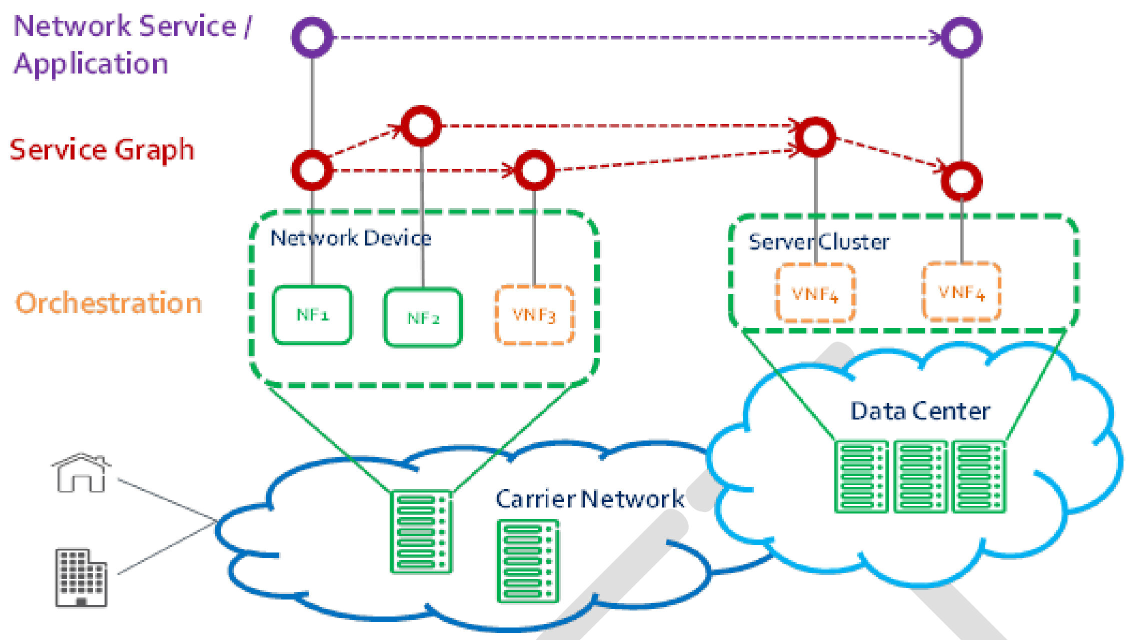


Figure 3.2: Flexible Service Chain

In the bottom, there are the physical resources, here split in two network segments under the control of one entity (single administrative domain). It consists of network devices in the carrier network part and servers hosted in data centre. The resources and the physical network functions (NF1 and NF2) are represented to the orchestration function.

The following list below tries to illustrate the process of ramping up a service and its associated service chain. The steps are as following:

1. A request for a service is transmitted.
2. The service request is decomposed into network functions (NF) with the help of some data bases (NF descriptions are stored in some type of catalogue). Between the network functions, the connectivity constraints are defined as well. The combined result is a specific implementation of a service graph, which defines the order of processing of network functions.
3. The service graph is forwarded to the orchestrator.
4. The orchestrator maps the service graph to the topology and available resources and defines the resource requests.
5. The resource requests are passed to the different hardware entities.

More details are provided in the following sections.

In general, the Flexible Service Chain should provide the possibility to customize in a very fine grained manner, the traffic forwarding on per user or application basis and on the other side provide cost advantages by processing flows using network functions only where necessary. In addition it is assumed that this flexibility reduces time to market for new functions and services as well.

It is envisioned that future orchestration and programmability framework within an administrative domain supports:

- Service graphs consisting of a large set of potential network functions like load balancer, DPI, certain gateways, NAT, parental control, content optimizer, etc.
- Interfaces providing programmability to the administrative domain's supported network functions (including interfaces for management systems, sharing of network functions between applications of the operator, etc.)
- Service graphs being shared between different applications and network services
- Abstractions of network functions available to application and network services
- Service graphs consisting of physical and virtual network functions
- Service graphs consisting of network functions hosted in data centre and provider network
- Service graph orchestration being aware of different available resource types including storage and processing
- Service graphs being able to add / remove network functions
- Migration of VNF between different locations / physical resources based on orchestration / optimization triggers / pre-sets
- Service graphs being specific to a single user, a group of users
- Awareness of forwarding rules based on different criteria like crossing real world borders, leaving the administrative domain, etc.
- Management of service graphs in carrier grade manner with support of OAM, fault analysis, etc.
- Service graphs being authenticable, authorizable and accountable

3.3 Use Case Group 3 - Network Service Chain Invocation for Providers

In order to support rich and flexible services with operational efficiency, we envision programmability within and across administrative entities. One can observe the following different entities:

- End-users: residential, business
- Infrastructure providers: Data centre providers that offer plain resources not services, network providers (access: fixed and mobile, backbone) that offer plain capacity not services, equipment providers
- Network operators: Network service providers that either own the network infrastructure that they use to offer services; thus, integrating the role of the infrastructure provider, or rent them from infrastructure providers

- Cloud operators: cloud service providers that either own the cloud/data centre infrastructure that they use to offer services, like storage, computation; thus, integrating the role of the infrastructure provider, or rent them from infrastructure providers, like data centre owners
- Service providers: Over-The-Top (OTT) service providers that offer services on top of network and cloud service rent by network and cloud operators. A special type is the Content Provider, who operates CDN on behalf of service providers.

Note that multiple combinations of the aforementioned entities may exist.

Based on the different entities involved in the interaction we envision different levels of programmability features (see Figure 3.3). For example, a retail provider shall be able to fully control all his/her assigned resources (for instance Network as a Services or Platform as a Service) while a residential user might only be interested in service selections or software as a service (SaaS) to instantiate arbitrary new network functions or reconfigure service parameters from time to time. Nevertheless, we expect that in all relations users may require specific service level agreements (SLAs), which they formulate according to their services (technology and topology agnostic ways).

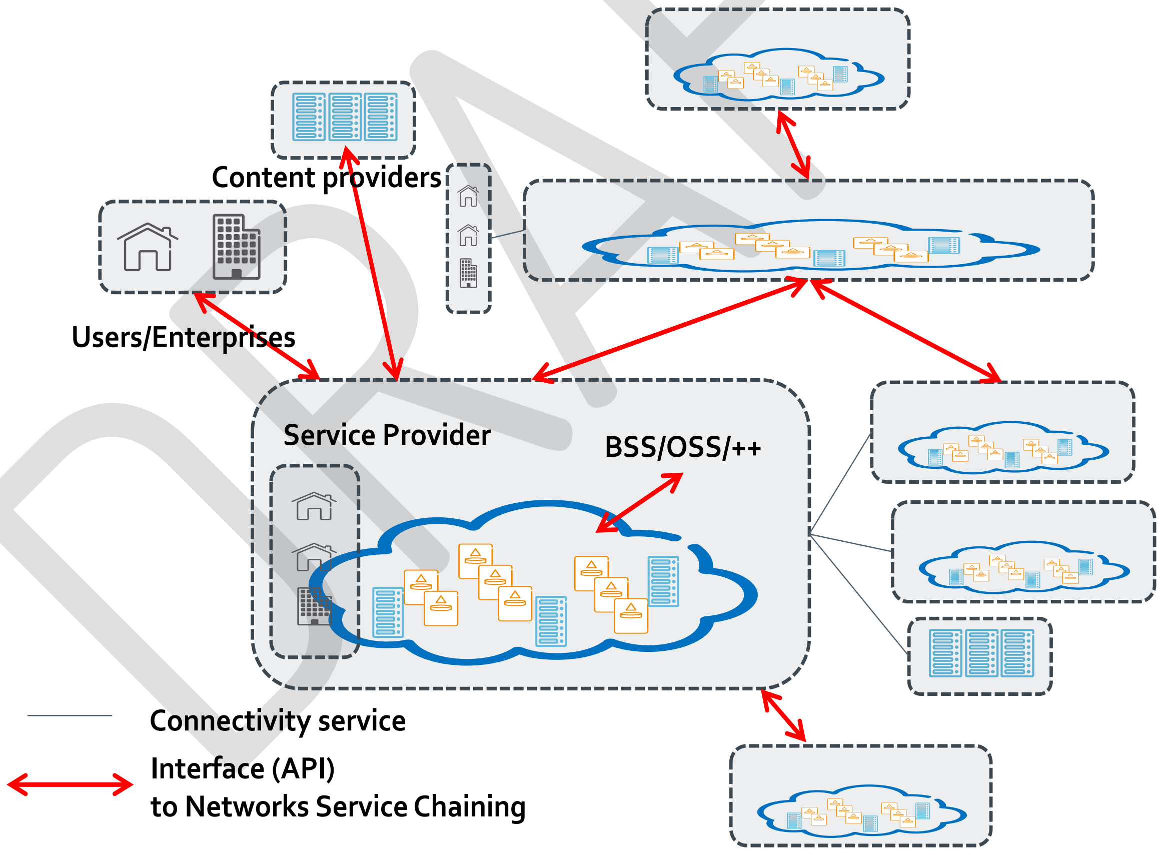


Figure 3.3: Programmability and business entities

The connections could be manifold with the infrastructure providers and / or network operators in the centre. The use cases cover connections between

- end-users and infrastructure providers or network operators

- end-user and service provider, where the service provider instructs the network operator
- network operators or infrastructure providers where communication across entities exists between peers on similar layers

Important in the multi-provider area is the definition of interfaces between the different roles. For the definition of the use cases, it is not important to completely define these interfaces but to describe their properties and desired behaviours of the interfaces in terms of dynamicity and supported procedures. Another important area is security between the entities on a platform.

3.4 Dedicated selected use cases

The manifold aspects representing the unified production environment are distributed over and touched by the various defined use case. , it was decided to concentrate the aspects in a low number of use cases only. It seems very beneficial for the further studies within the project to "condense" two of the most important aspects in UNIFY, the operation of virtual network functions on Universal Nodes and operational and management activities in a unified production environment in two additional dedicated use cases. The two use cases are:

- Elastic Network Function covering a generic Virtual Network Function on a Universal Node and highlighting the flexibility required in a UNIFY environment
- Secure & Content aware IP VPN covering the operational and management activities of an UNIFY environment and highlighting the intra and inter-domain activities especially for service maintenance and addressing end-to-end relations

3.4.1 Elastic Network Function

The objective of the Elastic Network Function use case is to demonstrate the different dynamicity and, more specifically, scalability approaches supported by the UNIFY architecture. This would serve as a basic building block for UCG1, and it will allow demonstrating the capabilities of Infrastructure Virtualization to support different approaches to scale a Network Function. In general, the Elastic Network Function would provide networking services and functions like firewall or routing that scale in real time to offer a more efficient resource usage, as well as the capability to dynamically adapt to overall requirement changes. In addition, the Elastic Network Function could be instantiated by a graph compromising of several different network functions and/or a number of instances of the same network function. One example application is shown in Figure 3.4, where an Elastic Router is used to connect the main office and the local branches of an enterprise. In order to cope with additional bandwidth requirements, the Elastic Router, for example, could add resources to the required instances and deploy a new path.

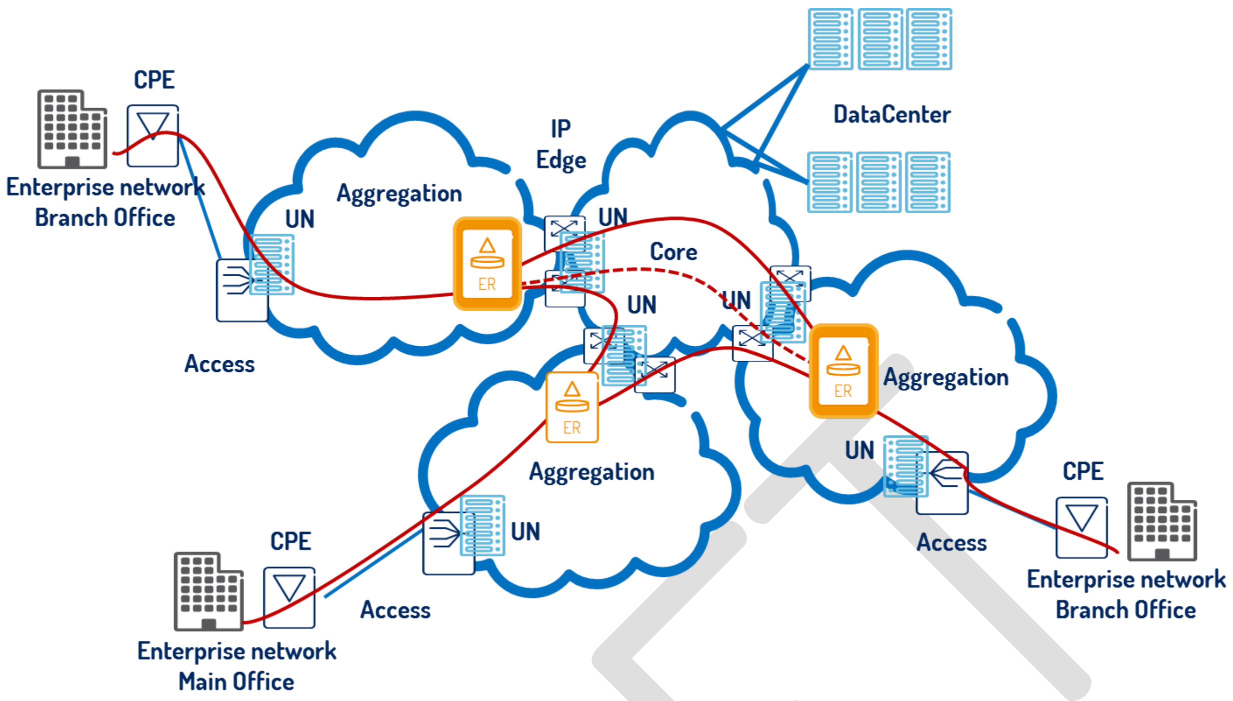


Figure 3.4: Illustration of the elastic router service

The possible approaches for managing the scalability have been categorized according to two different (and mostly orthogonal) criteria:

- How is scalability achieved
- Where (or by whom) is the scalability process managed

Even though the analysis is focused on the scalability (since it is the main aspect to demonstrate in the use case), it also applies to the whole (re)provisioning process, for which the scalability is a representative case.

3.4.1.1 Achieving scalability

How scalability issues can be addressed depends on the specific application. This aspect, though, will demonstrate the capability of UNIFY architecture to support new approaches in the design of virtualized network functions, such as following the SDN paradigm of decoupling the Control and Data plane components, thus providing independent scalability of each component depending on the application requirements. According to the element that supports the scaling and if the application logic of the scaled network functions (NFs) needs to be aware of the process or not,, the following categories of potential scalability mechanisms have been identified:

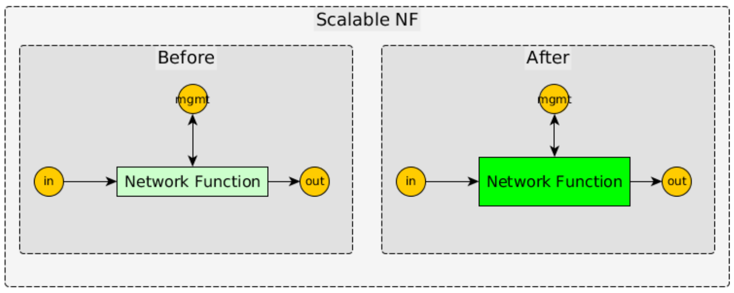
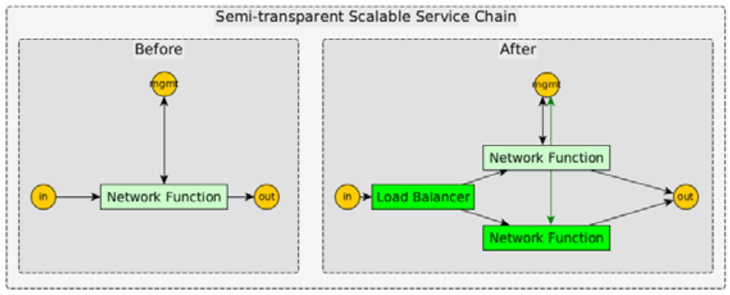
- Scalable Network Function (NF): the individual NF can be scaled up/down adding/removing resources to the NF (could be done live or as a VM migration process, depending on the application).
- Semi-transparent scalable Service Chain: the chain can be scaled out/in adding/removing instances of network functions in the graph and the scaled application logic is not aware that it is part of a bigger system. The main challenges of this approach would be:

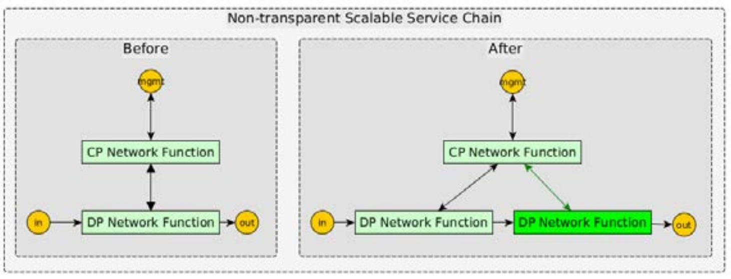
- Load balancing between the network function instances
- Consolidated configuration/updates of the network function instances (separate NF instances could have separate management interfaces).

- Non-transparent scalable Service Chain: the chain can be scaled out/in adding/removing instances of NFs in the chain and the NF application logic is aware that it is part of a bigger system. The main challenges of this approach would be:
    - Load balancing between the network function instances
    - Signaling or synchronization of the state between the NFs that need to be aware of the change.

InTable 3.4 the different categories are summarized. Of course, in the same chain there could be different elements, where each of them has a different support for scalability and also elements that support several types of scalability, and each of them allows to scale to a certain (and different) limit. For example, the Elastic Router from Figure 3.4 could scale either as a Scalable NF just adding more resources or as Non-transparent scalable Service Chain deploying new data path elements (more details in subsection 3.4.1.3). The example shown for the Non-Transparent scalable Service Chain is decomposed in a Network Function covering Control and Data Plane separately: the scalability is achieved by instantiating and removing Data Plane elements and the management part of the Control Plane must be notified of the change so it can control the new instance and split the load across all the Data Plane elements.

Table 3.4: Scalability options depending on the application

Scenario	Scaled Element	Network Function application logic aware	Example
Scalable NF	NF	No	 Scalable NF Before After mgmt in Network Function out
Semi-transparent scalable Service Chain	Service Chain	No	 Semi-transparent Scalable Service Chain Before After mgmt in Network Function out Load Balancer

Non-transparent scalable Service Chain	Service Chain	Yes	 Non-transparent Scalable Service Chain Before After mgmt CP Network Function in DP Network Function out

3.4.1.2 Managing the scalability

Unlike the selection of methods for achieving the scalability, which is application dependent, selecting the location where the scalability process logic is executed is a design decision, as each of the alternatives considered below opens up different possibilities and places different requirements on the application and the architecture. The alternatives defined are based on the location / responsible actor for monitoring the criteria/thresholds and triggering the actions for scaling.

The two main information flows required in order to manage the scalability are the monitoring of the trigger criteria and (re)provisioning request. The alternatives proposed are homogeneous according to the latter, while the former could be either supported by the UNIFY monitoring or by the inter-NF interfaces and be opaque to the UNIFY architecture (both alternatives can be considered sub-scenarios of each of the possibilities listed below):

- Managed from above the UNIFY architecture: the NF scalability management entity lays in the User/Application layer, monitoring the graph and requesting changes to scale up/down or out/in that would be similar to the original provisioning process and would traverse all the layers. An example is shown in Figure 3.5.

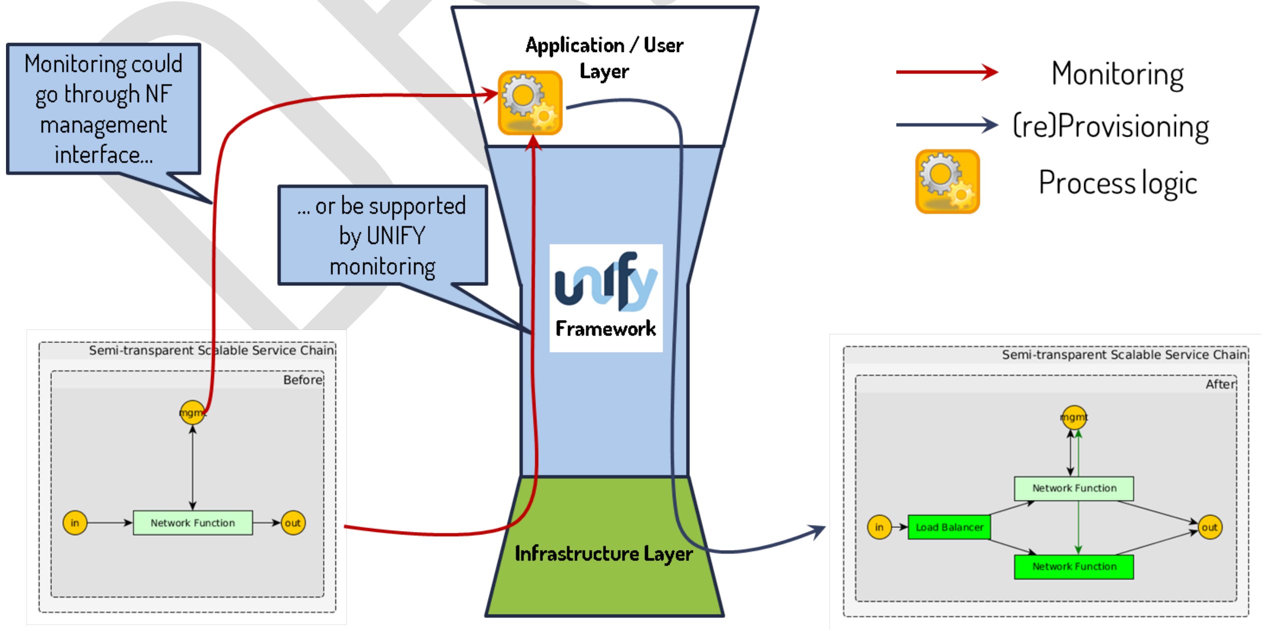


Figure 3.5: Scalability managed above the UNIFY architecture

- Managed by the UNIFY architecture: the scalability would be directly supported by the UNIFY framework. The graph would be required to contain both the criteria and thresholds to trigger the scalability, as well as the actions to be triggered. In this alternative the criteria must be monitored by the UNIFY architecture. An example is shown in Figure 3.6.

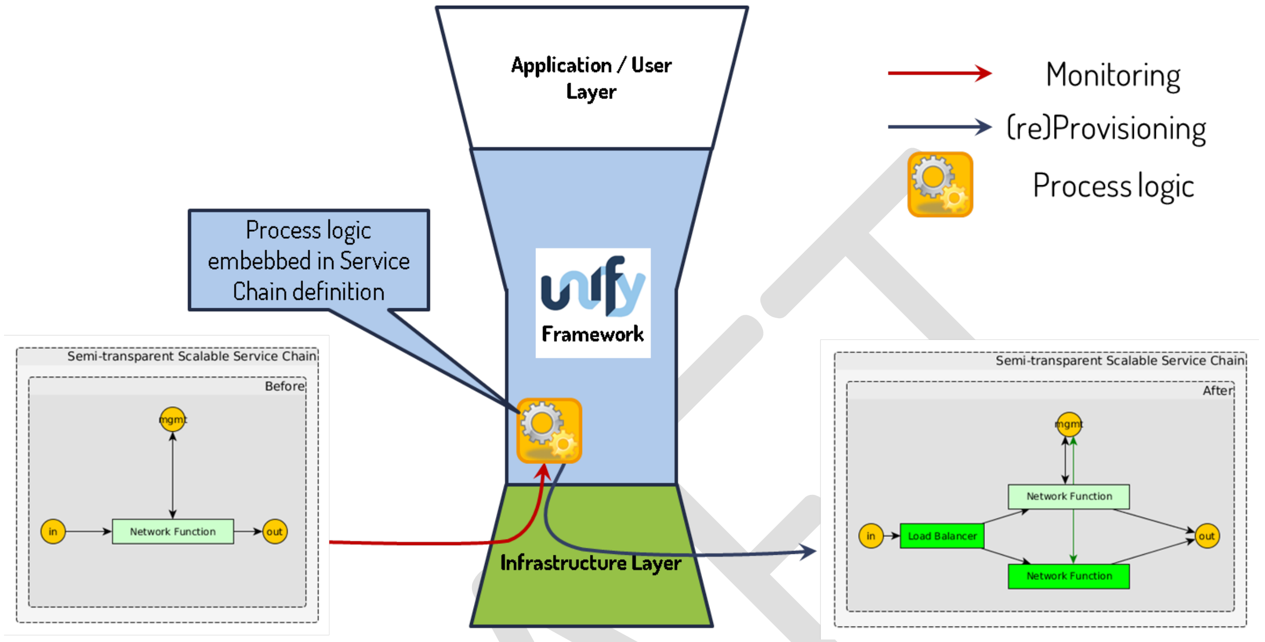


Figure 3.6: Scalability managed by the UNIFY architecture

- Managed with support from the UNIFY architecture: one (or more) NF of the chain (Control App) would monitor the criteria (either directly or supported by UNIFY monitoring) and request additional/fewer resources for the graph interacting directly with the UNIFY framework. The scope of the interfaces between the Control NF and the UNIFY framework would be limited to monitoring and resource requests, similar to the interface from the Service layer to the UNIFY framework. An example is shown in Figure 3.7.

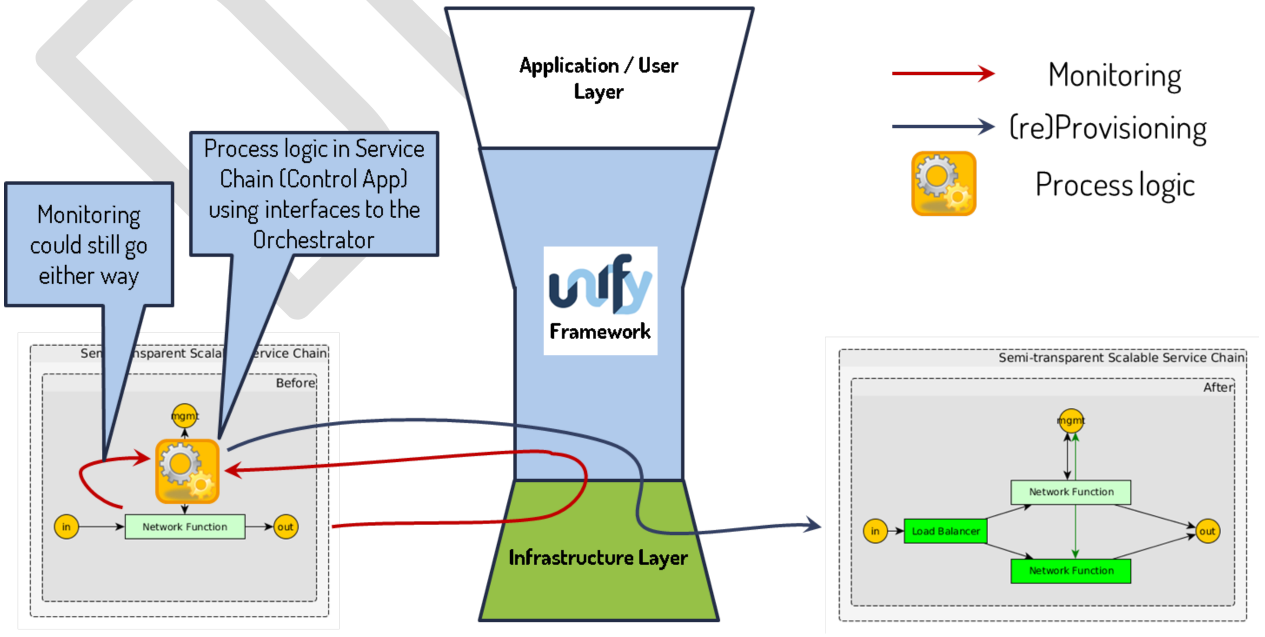

Application / User Layer
unify Framework
Infrastructure Layer
Monitoring
(re)Provisioning
Process logic
Monitoring could still go either way
Process logic in Service Chain (Control App) using interfaces to the Orchestrator
Semi-transparent Scalable Service Chain
Before
mgmt
in
Network Function
out
Semi-transparent Scalable Service Chain
After
mgmt
in
Load Balancer
Network Function
Network Function
out

Figure 3.7: Scalability managed with support from the UNIFY architecture

- Managed with separate application specific framework: in those cases where the information and capabilities provided to the Control App by the UNIFY architecture in the previous alternative would not be sufficient, the Control App could become something similar to the UNIFY framework with similar functionality able to exploit all the capabilities of the UNIFY architecture but restricted to a specific set of resources. The interaction would then be with the infrastructure (or through recursiveness, it might be another lower level orchestrator) so the scope of the interfaces would be augmented with respect to the previous option (Managed with support from the UNIFY architecture) to include the resource view (usage, topology, etc.). An example is shown in Figure 3.8.

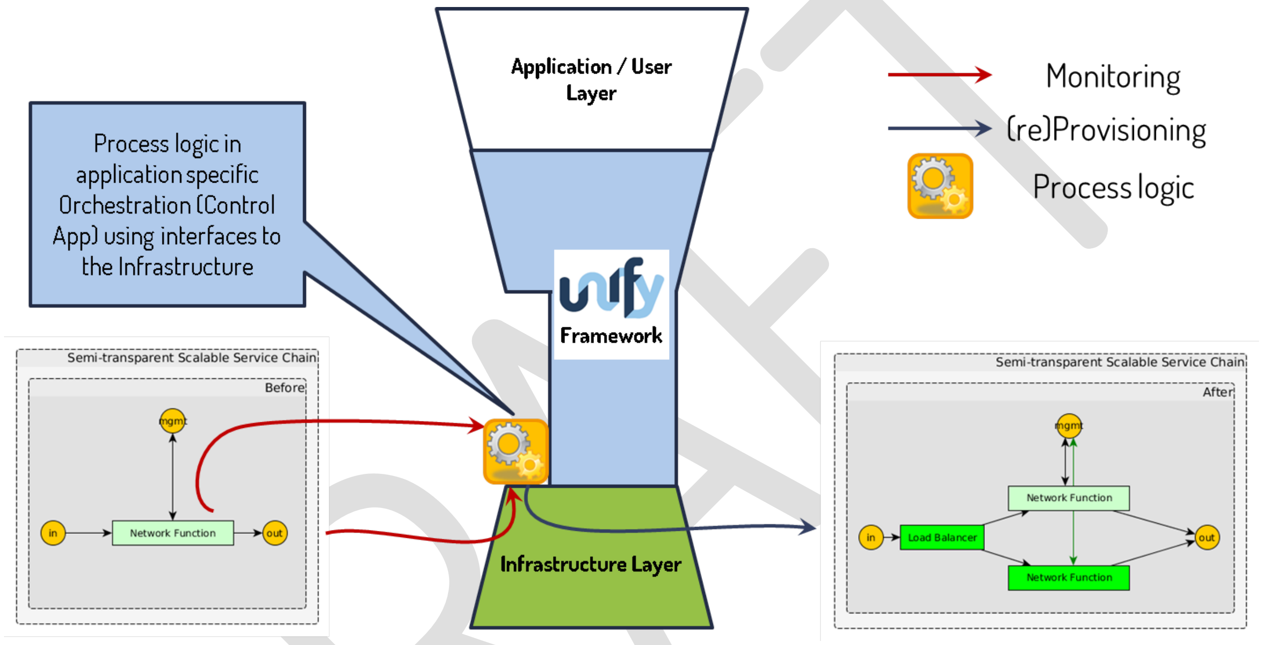


Figure 3.8: Scalability managed with application specific Orchestration

3.4.1.3 Scope of the Elastic Network Function use case

The proposed definition of the Elastic Network Function follows the SDN paradigm of decoupling the control and data plane, so it is composed of a Control Plane element that contains the routing logic and the management interface to configure the NF, and one or more Data Plane elements that implement the actual data processing following the instructions from the Control Plane.

This use case allows for different possible implementations of the aforementioned alternatives, both regarding how and where the scalability process is managed, it will be decided later on the project which will be finally implemented as proof of concept. Some possible options are:

- Scalable Network Functions, scalability managed by the UNIFY architecture: The graph contains the traffic thresholds upon which additional resources must be deployed/removed for the data plane part of the network function (the execution environment). The UNIFY architecture would monitor those thresholds and automatically assign/remove resources to the Data NF.

- Non-transparent service graph, scalability managed above the UNIFY architecture: The Control NF provides monitoring information about the current traffic being routed to the OSS/BSS which then can decide upon the need to instantiate or release resources to adapt the Elastic Network Function on demand. To achieve this scalability three elements are modified as shown in Figure 3.9, where an Elastic Router is used as an example of Elastic Network Function.
  - Data Plane elements are instantiated to scale up/out and released to scale down/in.
  - The Control Plane element is modified to add/remove interfaces to the Data Plane elements (at any time there will only be one Control Plane element to control all the Data Plane elements).
  - The traffic steering is modified to distribute the inbound traffic to the available Data Plane elements (thus providing load balancing capabilities) and direct all outbound traffic to the defined endpoints.

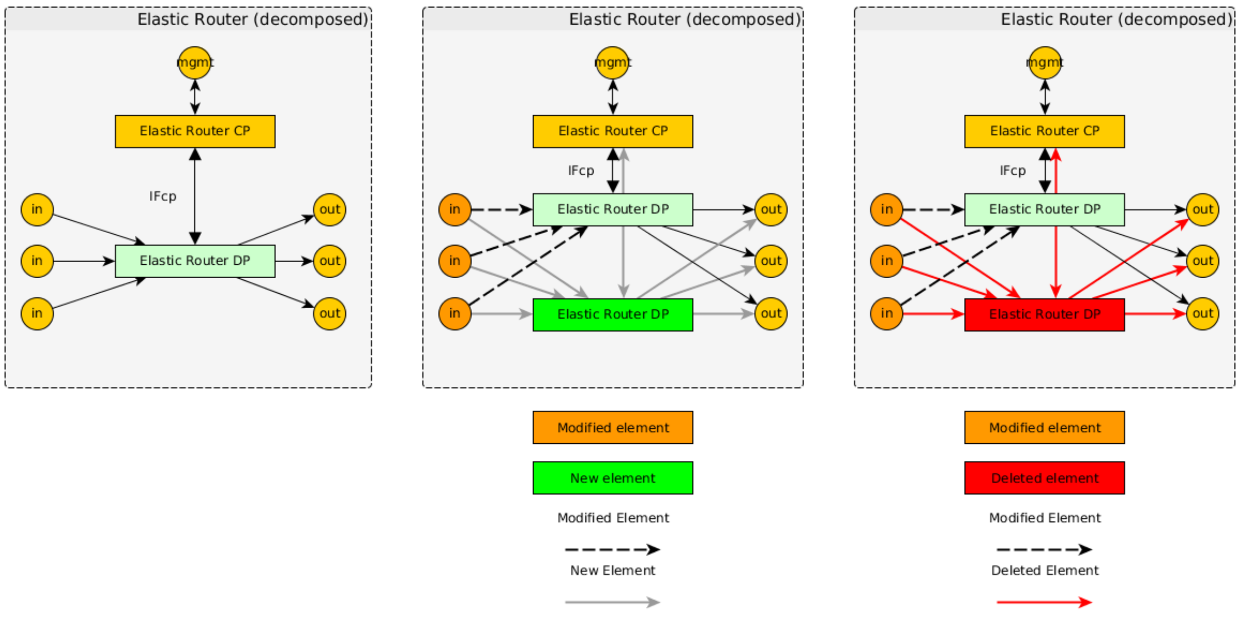


Figure 3.9: Example of Elastic Router scalability

The detailed analysis of this use case will be subject of future work and reported in upcoming deliverables.

3.4.2 Secure & Content aware IP VPN

The Secure, Content-Aware IP VPN is an example of a value-added service that a telecommunication carrier might offer on UNIFY infrastructure. Telecom operators offer VPN connectivity services to customers for a long time, using different technologies such as L2/L3 VPNs, SSL VPNs, etc. More advanced services, such as managed firewalls are also available to complement the VPN connectivity (for example [17] or [18]). Specialized companies that build data centres across the world are also competing in this market (for example [19]), while offering additional opportunities such as managed intrusion detection systems. It is natural for telecom carriers to enhance their service offering and leverage their superior control over connectivity between enterprise sites within their network. Building modular services, on which optional fully-managed components may be added and removed with simple actions in a dynamic manner, provides opportunities for carriers to compete better in providing managed services to enterprise customers.

Compared to a classical IP VPN service, the carrier provides value to enterprise customers by embedding SSL accelerator functionality, a malware and intrusion detection function as well as private content delivery network (CDN) support and an elastic router (ER). The carrier has the ability to consolidate virtual appliances throughout their network infrastructure thus aiming to inherently reducing the total cost for providing the service compared to a dedicated deployment.

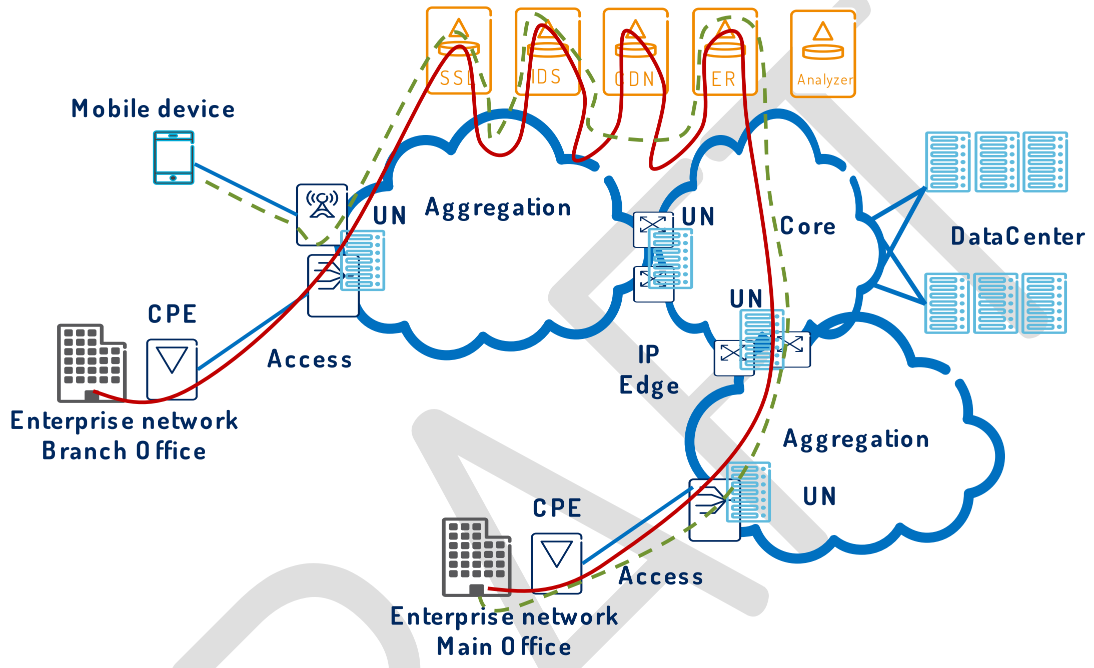


Figure 3.10: Illustration of the secure, content aware IP VPN service

By offering these functions as part of the service, the carrier enables the customer to reduce costs by not having to purchase, deploy and operate physical or virtual appliances. It also enables better quality of experience by being able to locate instances of these functions closer to the mobile worker, thus eliminating additional latencies that might have been incurred by connecting through a corporate gateway located at the main office but needing to access data from a branch office storage system, for example. Such latencies are important, for example, for enterprise representatives that perform co-creation activities at their respective customer sites, need to conduct interactive whiteboard sessions with in-house experts during trade fairs, or simply make use of an office rental spaces closer to their homes than the corporate offices.

The service has an inherent dynamic aspect in that policies associated to it allow the enterprise to increase the resource usage for a relatively limited time interval, in the order of hours, with no prior notice to the carrier. This is in order to facilitate nightly backups for disaster recovery (an activity that could easily be scheduled by the IT personnel), as well as massive demonstrations or interactions with customers during trade fairs (an activity that would require interaction between the trade fair organizing team, the internal IT support and the carrier contacts or a self-service portal if scheduling activities would be performed in advance). This enhances significantly the flexibility of the enterprise. The increase in resources is supported mainly through a) an elastic router function (as introduced in section 3.4.1) located closer to the main office—where traffic aggregates are the largest—as well as b) an elastic cloud environment hosting network and application service blocks like CDN or conferencing systems and c) a programmable transport network providing dynamic, on-demand transport services among all service blocks. An example realization of such an environment with the high-level service blocks is depicted in the Figure 3.10, with Universal Nodes at the network edge, cloud environment hosting the services and an SDN-enabled transport network including also a packet-optical DWDM circuit-switched transport network segment controlled by an SDN controller (for example an ODL controller equipped with the circuit-switched services). The SDN controller enables the carrier to efficiently support service provisioning and resource optimization through dynamically (re)establishing links (e.g. WDM, Ethernet, IP, etc.) between service endpoints and the core network or between endpoints and network functions. In addition, traffic forwarding decisions are optimised with the help of an analyser function allowing network service chains (or virtual network function chains) to forward traffic through necessary elements only, e.g. the mobile access requests a connection to a colleague at the main office with security but not with content access.

Other examples of dynamicity aspects associated to this service include:

- The intrusion detection function (IDS) is refreshed with new attack vector signatures daily, as soon as they become available through updates issued by the function developer company. New and improved detection algorithms are to be deployed several times a year. New content management features and bug fixes are also available several times a year, including critical security updates that have to be applied in advance to scheduled maintenance windows.
- Forwarding decisions are modified during runtime, meaning that traffic identified as malware can for instance be quarantined and made to bypass the content delivery system. In today's enterprise networks, it is notorious that compromised documents attached to email messages are used as vectors for attacks. Not distributing the document further through the corporate content delivery system is one of the measures that could be taken in order to reduce the risk of infecting additional users.

The carrier targets to be able to deploy the service within hours from the arrival of a request from a new customer. This is, of course, provided that the basic connectivity to the customer site exists and thus truck rolls are not needed. By having the capability to fulfil the order in such a rapid manner, the carrier increases their ability to compete with over-the-top providers of virtual appliances that make use of public or private cloud resources.

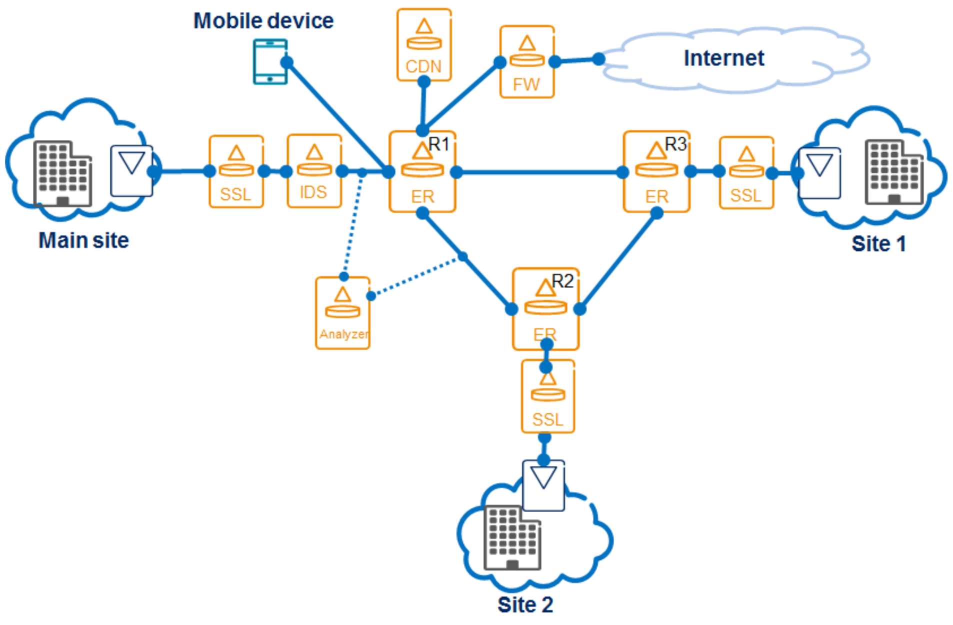


Figure 3.11: Illustration of forwarding graph

The forwarding graph associated with the service is depicted above in Figure 3.11 with the sequence of VNFs. Beside the high-level service elements depicted in Figure 3.10 already, a traffic analyser function, for classification of traffic and management of forwarding decisions for service chains, as well as an Elastic Router is added. Connectivity between the elements of the service graph is supposed to be ensured by Layer 3 routing functions which are not included in the graph. Such a forwarding graph could be supported by the following service functions mapping as reported below in Figure 3.12.

Depending on the implementation options, service levels specifications agreed upon and the transport network availability, VNFs could be shared by multiple customers or utilized exclusively for one particular customer. The granularity of association within the service chain is given by traffic originating from a customer-premises attachment point to the service. It should be noted though that some flows might be made to bypass certain elements in the chain – for example, flows that are considered to carry attack vectors will avoid passing through the Content Delivery function. In general, the exact forwarding decision may depend on the type of suspected traffic, so that certain known attack vectors are dropped immediately, while others might be allowed to the destination but not sent through the CDN. All different variations will be supported through the programmable transport network.

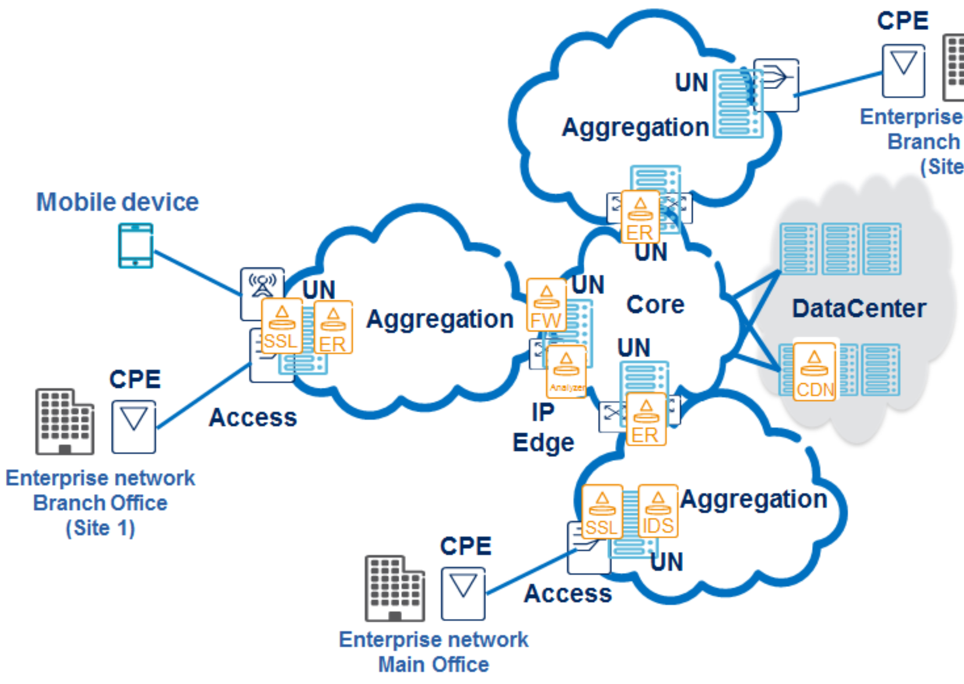


Figure 3.12: Service functions mapping in the network

The VNFs are expected to be executed on Universal Nodes located in the provider's infrastructure, with the possible exception of the terminating SSL encryption and decryption functions which might be executed on provider-controlled equipment placed at the customer premises. This depends on whether the connection between the customer premises and the closest provider PoP could be considered secured or not. If the connection could be considered secure (for example, in case of a single optical fibre not shared with other customers), the SSL encryption and decryption could be placed on provider premises. Otherwise, they would be based on customer premises equipment controlled by the provider.

Virtual network functions performing monitoring of the capabilities are not shown as part of the service chain because, on the one hand, they are not available directly to the customer and, on the other hand, a multitude of such capabilities needs to be deployed for monitoring the use and performance of resources at network, compute and storage levels.

Based on the use case described above, a related proof of concept activity could be executed in a subsequent step, to demonstrate the credibility of key aspects and solutions.

This proof of concept activity should be based on the main ideas expressed by the use case and reflect its essence, even if some of the detailed implementation aspects could be somehow dictated from the characteristics of the concrete credibility environment implementation selected. In particular, given the service concept to be demonstrated, the specific service functions to be used could be determined by the real availability of means and resources. Examples of service functions that could be used for demonstrating the use case are: Deep Packet Inspection, Firewall, Caching and URL Filtering (but others could be used as well anyway). Although different in terms of the individual functionality offered, these service functions can be arranged together anyway in order to create service chains for assessing the use case.

Independently form the specific service functions to be used for the proof of concept implementation of the use case, it would be strongly necessary to consider in the credibility environment the implementation of a service chain scenario in which aspects like dynamicity and flexibility (e.g.: fixed or variable shape with possible changes to the data path for a given traffic session/flow and possible asymmetry in data forwarding) and virtualization of service functions are addressed. Moreover, in order to reflect the key ideas of the use case, in accordance with the UNIFY project goals, centralization of some service functions in a Data Centre environment versus distribution of others should be implemented as well as the coexistence of physical service appliances and virtualized service functions in service chains, in order to evaluate also possible issues in a migration perspective. The proof of concept environment could be set up either considering only one test bed or more geographically distributed test beds.

Finally, with reference to the choice of which technical solution to use to implement the use case and create the proof of concept environment, a prior activity of investigation and analysis of the solutions being under exam and study by standardisation groups and available from vendors should be necessary.

4 Requirements

This section reports different groups of requirements relative to the UNIFY architecture framework. The first group is reported as "general requirements on the framework and architecture" and includes business, high level architecture and operational requirements. The other groups of requirements reported are: requirements for service programming and orchestration, requirements for Service Provider DevOps and requirements for Universal Node architecture and evaluation. An important source for the definition of the requirements was the ETSI ISG on Network Function Virtualisation (see section 2.3) and therefore references to these dedicated requirements are included (marked with [XY]).

4.1 General requirements

4.1.1 Business requirements from the operator side

The business requirements reported below deal with the main aspects of the general environment that will be addressed by UNIFY. These requirements are of more general nature, but define the scope of certain implementation aspects. In general, it is expected that UNIFY solutions will play an important role in typical carrier, data centre and private networks. Therefore these four requirements try to cover the different aspects.

Req 1-1 The framework shall allow multiple operators / providers to interwork with each other.

It is expected that the freedom for implementation will restrict the requirements to a more general level. For example considering requirement 1-1, the support of multiple operators for forwarding could be enabled in different ways with slicing of the network with pre-configured available bit-stream products, a split of the available address spaces or with full virtualisation. Also the multi-operator support of network functions is another aspect. Corresponding ETSI Requirement is [Gen.1].

Req 1-2 The architecture shall support different users and user groups.

This requirement covers the aspect that the users and their service could be of different nature. In addition, some users could be grouped like all users of an enterprise network. A first set of users is discussed in section 3.3 and potential roles identified in the architecture in section 6.6.

Req 1-3 The framework shall allow reasonable interfaces in between stakeholder groups.

In order to reflect the state of the art (section 2) and the use cases (section 3), requirement 1-3 is supposed to reflect the demand for collaboration in between different players, something which is difficult to implement today. A first preliminary set of players is introduced in section 3.3 and roles discussed in section 6.6.

Req 1-4 The framework shall support interaction with legacy equipment.

Introduction of completely new architectures and components is only possible in very special cases. Most often an interworking with existing environment must be respected (requirement 1-4). The corresponding ETSI NFV requirement is [Mig.1].

4.1.2 High level architecture and framework requirements

The high level requirements are relative to the architecture definition. These requirements are outlined underneath.

Req 1-5 The architecture shall support extensions.

Requirement 1-5 is the core and very representative of the UNIFY main idea, providing a framework and architecture which allows extensions and support, e.g.

- Introduction of new services
- Additional infrastructure components
- Development of controller
- Introduction of new protocols.

Req 1-6 The architecture shall allow migration in-between different technologies and releases.

With requirement 1-6, it is addressed that changes of and in the architecture happen quite frequently. For example software components are exchanged for performance or security improvement, new hardware is added to the deployment, etc. Therefore it is demanded to be independent of used programming techniques.

Req 1-7 The architecture shall be scalable according to desired use cases.

Scalability as demanded in requirement 1-7 is typically hard to define. In subsequent work of the definition of the programming framework (D3.1), quality and performance indicators will be subject of research and will be developed and defined more in details. In addition, use cases to be implemented (see section 3.4) will require scalability targets in terms of users, packet throughput, etc. for verification. A detailed list of parameters will be documented in D2.4.

Req 1-8 The framework shall support the virtualisation of all functional elements.

The designed framework has to support all kinds of functional elements like forwarding, but also hardware acceleration features, network functions, etc. as demanded in requirement 1-8. This does not necessarily have to be implemented by UNIFY, but the framework should be open and extensible to do so. The corresponding ETSI NFV requirement is [Gen.1] and [Port.2].

Req 1-9 The framework shall target standardized interfaces and components.

Finally in requirement 1-9, it is addressed that the standardized interfaces aim for openness and multi-vendor support of the architecture. Other aspects include the definition of containers of components and their respective management. The corresponding ETSI NFV requirement is [Port.1].

4.1.3 Operational requirements

Below are reported the main operational requirements expressed from an operator's point of view.

Req 1-10 Required information must be descriptive.

Requirement 1-10 means that the information exchange should be available in a way that the system and its interfaces understand or at least can translate the information. In addition, it is helpful for trouble-shooting and operation if the information can be read by humans, that is something which command line interfaces normally do not provide.

Req 1-11 Configuration and monitoring data must be made available and documented.

Again, requirement 1-11 captures the demand for support of operation to access actual and historic configuration and monitoring data for restoration and backup solutions.

Req 1-12 Configuration and operation of network and services should be performed automatically and autonomously.

Automation as detailed in requirement 1-12 is one of the desired behaviours of the UNIFY solutions. Supporting more services and hardware means some higher levels of complexity, which in turn might demand higher automation from the UNIFY solution. Nonetheless, requirements 1-10 and 1-11 must be taken into account at the same time.

Req 1-13 Configuration of network and services should be performed in a consistent process.

Requirement 1-13 means that the configuration steps of the architecture are finally implemented and executed without inconsistency.

Req 1-14 Configuration of service chains should be supported.

Requirement 1-14 refers to the fact that having an increasing number of services combined results in efforts to design service chains as detailed in state of the art (section 2.7.1). This should be supported by the UNIFY architecture as well.

Req 1-15 Validation of configurations across the service chain and its elements should be supported.

Overall, requirement 1-15 demands support for validation of possible network and service configurations. This helps for example in the evaluation of service combinations by customers.

Req 1-16 Link resources should be efficiently used and traffic should pass through necessary elements only.

The requirement 1-16 defines a very important aspect in terms of costs of providing a network. Typically, interfaces are the major cost point of networks and they should be limited as much as possible. In addition, service chains have been difficult to implement in the past and current developments can optimise the bandwidth and therefore interface demand. One such effort is service function chaining discussed in IETF and briefly summarized in section 2.7.1.

Req 1-17 Service chain elements should be transportable across the network in all network domains / segments (access, transport, cloud, data, virtualization, etc.).

It might be required to move network functions around as detailed in requirement 1-17. For example a customer might request that a service function hosted in the data centre be migrated to his home environment. So, there are two aspects to be considered: the network function should be transportable itself and on the other side if the network function must be adapted or replaced by a more suitable one configuration data should not be lost.

Req 1-18 Operator policies have to be respected.

In certain cases, operators might want to implement policies for defining the behaviour of the system as expressed by requirement 1-18. For instance resources should be restricted to customers, network elements should be used for certain services only, etc.

4.2 Requirements for Service Programming and Orchestration

Supported by state of the art and use case analysis, these requirements are driven by programmability and orchestration aspects that will be documented in D3.1. Each requirement is applicable on different, already identified resource types like the Virtual Network Function (VNF), the service graph – the combination of network functions and links – or on the resources. The requirements have been sorted according to their importance, starting with the highest level of importance.

Req 2-1 Descriptions of virtual and physical network functions as well as forwarding information must be available for programming service graph.

With reference to requirement 2-1, the detailed descriptions with languages and information models are under development and will be reported in deliverables D3.1 and D3.2. Currently it is assumed that virtual network functions will contain information about computing and storage resources, the physical network functions could have their own descriptions and forwarding information must be descriptive in a way that connectivity between network functions is possible. An example of forwarding information is OpenFlow which should be supported.

Req 2-2 Configuration data must be conveyable and semi-transparent.

Requirement 2-2 describes the fact that abstraction and virtualisation could force modifications of address spaces, protocols or specific configuration data.

Req 2-3 OpenFlow compatible traffic steering definitions should be supported.

OpenFlow is one of basic abstractions for networks today.

Req 2-4 Performance indicators of service graphs and network functions should be supported as part of the descriptions.

It is of major importance to define the quality of experience and service from a user's perspective and translate them into machine understandable parameters. In addition, the quality must be observed, analysed and react in case of breaches. In order to automate these complex activities it is of importance to add this information to the service graphs and network functions.

Req 2-5 Preferences, restrictions and pinning of service graphs and (virtual) network functions should be supported as part of the descriptions.

Requirement 2-5 reports that forwarding across certain domains could be limited or a physical appliance should be preferred over virtualized network functions. Another issue is that certain (virtual) network functions can run on specific hardware only and it must be possible to describe these interdependencies.

Req 2-6 Optimisation triggers and goals of service graphs and (virtual) network functions should be supported as part of the descriptions.

Optimisation addressed by requirement 2-6 could be performed based on different counters and triggers. E.g. the description contains certain triggers for scaling resources dedicated to an environment, than there must be appropriate understanding how to deal with this.

Req 2-7 Integration/instantiation of new virtual or physical network functions must be possible.

This is adding another detail to the requirement 1-5. It must be possible to instantiate virtual network functions on compute resources on request in order to fulfil the service requests. The same principle applies to physical network functions.

Req 2-8 Steering traffic to, from and between (virtual) network functions must be supported.

This requirement is a consequence of splitting services into smaller pieces and potentially distributing them throughout the footprint. So the traffic between them needs to be described and finally configured in the network forwarding and processing entities.

Req 2-9 (Virtual) network function and traffic steering demands and configurations must be modifiable.

The requirement 2-8 detailed the demand to configure the steering, there as requirement 2-9 request the possibility to modify the already configured steering. This requires support and continuous observation of network forwarding and processing entities.

Req 2-10 Triggers, subjects and constraints should be taken in consideration in the programmability framework.

According to requirement 2-10, the programmability framework has the task to support the resource organisation and configuration. There are different possible triggers demanding modifications, e.g. a control plane could request modifications for forwarding and network service chains, another one could be an optimisation trigger which wants to modify the dedicated resources or also a typical case could be the failure of a certain component which could require the modification of a service graph. In addition, there could be requests at different architectural layers, for certain parts of the service graph only, etc. The corresponding ETSI NFV requirement are [Port.3], [Elas.2], [Elas.3] and [EE.2].

Req 2-11 **Resource availability exposure should be provided at different granularity levels (service performance parameters, location, capacity, etc.).**

Moreover constraints are of different types as well and are defined by network functions, forwarding capabilities or even administrative decisions. They are part of the descriptions as expressed by requirement 2-11 and must be supported in the installation and configuration. The corresponding ETSI NFV requirement is [Elas.3].

4.3 Requirements for Service Provider DevOps

This section presents the requirements derived from the points developed within the deliverable document D4.1 about the use cases, state of the art and all the aspects regarding integration of service provider based DevOps (SP-DevOps). Further details will be documented in D4.1, including more explanations of requirements realizing SP-DevOps with the UNIFY architecture.

In functional terms, the SP-DevOps concept includes requirements regarding the main topics of Observability, Verification, and Troubleshooting. In the following definitions and examples of these three topics are given. Additionally, requirements regarding support of developers of VNFs have been considered as well.

In control theory, a system is called observable if we could reconstruct its internal state based on outputs that can be measured seemingly simultaneously. In a real complex system, such as a telecom network composed of tens of thousands of physical nodes on which millions of software processes are executed, continuous experimental access is limited to only some of the variables that describe internal system states. Such a system is thus only partially observable. The term Observability is used to collectively refer to methods that attempt to measure or estimate metrics or parameters and based on them determine particular system states in the UNIFY production environment. Examples of metrics include network delay, jitter and packet loss, processor utilization by a particular process, container or virtual machine; parameters include buffer occupancy, number of flows active, number of containers deployed on the same server; system states include link-level forwarding for a particular flow, network-level forwarding according to a particular routing protocol, the stage in the orchestration process reached by a particular NF-FG at a certain point in time during deployment, whether a particular container is being migrated or not, etc. Moreover, Observation of the performance in the network, of computing and storage resource as well as services is crucial for addressing flexible instantiation of service chains and to provide dynamic scaling of service components relative to aspects like the concentration of users and changing network conditions. The information provided by the monitoring capabilities is also important to ensure performance specifications and SLAs. Monitoring capabilities are envisioned to be performed by monitoring functions. In the deliverable document D4.1, monitoring functions are defined to consist of one or several observability points instantiated on one or several nodes together with a control plane component for analysis and control of lower-level monitoring operations towards the observability points. In turn, an observability point operates in terms of a node-local control and data plane components for local analytics and measurement purposes.

The term Verification is used to collectively refer to approaches that compare and contrast expected and detected system states in the UNIFY production environment. The expectations are based on pre-defined representations of the system states under investigation, while observability provides the detection of the actual state. Examples of such system states include availability of cloud infrastructure, compliance with security policies, the existence of forwarding loops or whether a certain node is reachable or not. The verification tools offer possibilities to debug service components during the design phase to ensure intended functionality, as well as ensuring resource availability and verification of parameters settings based on the specifications of service graphs and components. During the deployment phase, the validation and verification tools may need to operate within specified bounds (for instance time limits) to ensure rapid service-chain (re-) deployment.

The term Troubleshooting is used to collectively refer to techniques that correlate and filter information collected from different entities within the UNIFY production environment with the purpose to identify a particular erroneous situation. The correlation refers to collecting and assembling together data along a set of rules or descriptions associated to the particular erroneous situation to be investigated. The envisioned troubleshooting mechanisms will need to operate on the different levels of the architecture and should be able to utilize observability points and verification tools as part of process to determine the cause of a fault or performance degradation. The outcome of a troubleshooting session may be used to notify the service / VNF developer, and/or as input to the resource control/service orchestration parts for resource management to ensure continuous service delivery (for example trigger re-optimization of resources). Such data could include, for example, the content of the flow tables within a virtual switch and the erroneous situation under investigation could be the non-forwarding of a particular traffic flow. To properly perform troubleshooting, the DevOps framework should in general have the possibility to retrieve current and historical information from physical and virtual entities. Troubleshooting may include for instance the identification of the cause of performance degradation to rule inconsistencies or physical problems in the infrastructure.

In the following, the key requirements are identified and explained. Further detailing will be documented in D4.1.

Req 3-1 The UNIFY architecture must support capabilities to develop and test components.

Requirement 3-1 aims to facilitate increased service velocity towards customers by continuous deployment and integration practices. This reflects DevOps principles by making it possible to develop and test VNFs and service or network graphs in production-like systems. These capabilities can for example include creation and isolation of resources slices (computer, network, storage) and isolation or special treatment of downstream traffic from development virtual network functions (VNF). A complete list will be documented in D4.1. Moreover on the Universal Node there should be the support of sandboxed execution environments for non-trusted network functions and for validation purposes.

Req 3-2 The UNIFY architecture must support advanced monitoring capabilities.

As expressed in requirement 3-2 observability components go beyond the collection of statistics and simple counters from resources. As an example, it is envisioned that observability components provide scalable and detailed performance metrics. These components can in turn require support from the UNIFY architecture and infrastructure, for instance in the form of: ability to perform packet manipulation (like adding timestamps, mark a certain packet for monitoring, etc.) or to implement simple decision function on virtual resources (thresholds for notifications, aggregation of events, etc.). A list with more detailed examples will be documented in D4.1. The corresponding ETSI NFV requirement is [Res.6].

Req 3-3 The UNIFY architecture must support automated integration of monitoring, trouble-shooting and verification capabilities.

Requirement 3-3 mandates programmable interfaces towards monitoring and verification capabilities on all architecture layers to support automation of operational processes. Automation of operational processes will allow adaption and coordination of workflows for service design, development and operations teams.

Req 3-4 The UNIFY architecture should be able to react according to reports and notifications generated by observability and verification components.

Requirement 3-4 specifies the observability and verification components which will asynchronously generate reports and notifications (for instance the exceptions with respect to pre-configured or adaptively set thresholds and limits, such as changes, performance degradations, SLA breaches, etc.), that the UNIFY architecture will need to provide means to react in a way to mitigate the reported problem. This could be e.g. to forward the notification through a service layer interface to the application logic, or to trigger an appropriate function within the orchestration or service layer (e.g. a scaling function, or a function providing some sort of re-optimization through changes in VNF placement or rerouting).

Req 3-5 The UNIFY architecture must provide interfaces to access observability metrics of application and network services and their associated components in a desired timely manner.

Requirement 3-5 should support dynamic service optimization by mandating interfaces that propagate monitoring information in a desired timely manner: from close to real-time fashion up to hours. This means, for example, that the monitoring capabilities of the OpenFlow protocol could not be sufficient for matching real-time definitions. This may for instance require counters in the data plane capable of performing arithmetic operations such that flow counters can be modelled at higher sampling rates compared to what is possible today.

Req 3-6 Monitoring capabilities must allow for dynamic installation, activation and deactivation on request based on service provisioning and operational aspects.

There could be two different general sources calling for monitoring capabilities. First the analysis of service request during the provisioning process detects a demand to monitor certain traffic or, second, during the operation is identified that a monitoring function is required and dynamically integrated or activated.

Req 3-7 **The UNIFY architecture and its components should provide monitoring information in suitable level of detail and granularity according to the needs of applications or functional blocks within the architecture.**

In requirement 3-7 the demand for appropriate information for different abstraction and virtualisation needs is covered. For example, information might be needed per application, user or even finer granular element in the architecture. In addition, resources could be required in different granularity levels as well, like the specific state of port or the consumption or a more coarse granular level the state of a partnering data centre. The corresponding ETSI NFV requirements are [Perf.3], [Perf.4], [OaM.10] and [OaM.14].

Req 3-8 **The UNIFY architecture must support automated verification of services and representations.**

The interfaces supporting automated verification of services and forwarding graphs of requirement 3-8 must work on all architectural layers (i.e. levels of abstraction), e.g. formal verification of definitions and configurations with respect to validity and inconsistency. To support verification functions, relevant pieces of information must be provided by the UNIFY architecture

Req 3-9 **The verification functions should operate at service design and (re-) deployment time.**

Before the service request is finally installed, it should be verified that for example the type of resource is available, that the user has access, etc. In addition, this operation should be performed any time a modification is required.

4.4 Requirements for Universal Node architecture and Evaluation

This section presents the requirements which are derived from the developments on the Universal Node concept and architecture, documented in D5.1 and subsequent deliverables.

The Universal Node (UN) is a network node, based on COTS hardware that can execute network functions (packet processing) as well as more traditional cloud workloads. The UN integrates with the UNIFY orchestration and enables flexible traffic steering to the running functions or applications. The UN is optimized for high-throughput packet processing from the network interfaces to the network functions but also allows deployment of unmodified traditional applications with less stringent packet processing requirements. An UN supports various execution environments for NFs and applications, using virtualization or not, thereby providing different combinations of isolation, performance and ease of development and deployment. The UN concept may be scaled from small systems that could be deployed at customer premises to much bigger systems that could consist of multiple (heterogeneous) blades or even multiple (heterogeneous) interconnected chassis. The initial focus is however on single server UN.

The requirements have been derived from the different use case groups described earlier, but in particular from the "Virtual Residential Gateway – BNG" use case in group 2 "Flexible Service Chaining" and the Elastic Router example as implementation of the Elastic Network Function. Both could serve as a basis for the development of the Universal Node architecture.

Req 4-1 The UN must contain virtual switching functionality that allows switching traffic between the physical ports and the virtual ports of the VNFs present on the node.

This is a base requirement for the operation of a UN. Virtual Network Functions need to be attached to the physical forwarding and processing entities and therefore a virtual switching functionality is required.

Req 4-2 The UN must allow the network controlling entity to dynamically program the virtual switch(es) to enable and control the traffic steering.

Related to requirements 2-8 and 2-9, the UN must support this as well and allow the traffic steering.

Req 4-3 The UN must support the deployment or deletion of virtual network functions on request.

The requirement 4-3 describes the required support of virtual network functions in the UN. These virtual network functions could have different types of implementations. This could be for instance full-fledged Virtual Machine running in a fully virtualized execution environment in which case the VNF execution image includes a guest Operating System in addition to the VNF application itself, or as lightweight VNFs which will use the same base execution environment as the vSwitch or as non-packet-processing VNFs (for example distributed control plane apps). The organization of the deployment is part of further research.

Req 4-4 The UN shall allow scaling up/down of resources allocated to a VNF on request.

Triggers for scaling resources dedicated to a VNF as stated in requirement 4-4 could be triggers configured in the data plane or from an orchestrating entity, user interventions or from the application itself. Important is the organization and authorization of such triggers as well.

Req 4-5 The UN must provide an interface to describe its capabilities and available resources.

The capabilities and resources mentioned in requirement 4-5 could be CPU and memory resources, network resources (like ports, bandwidth) or available accelerators that may be accessed directly by specific VNFs.

Req 4-6 The UN must allow the configuration and operation of observation points on the node.

Requirement 4-6 will be further detailed according to requirements and developments of the monitoring and observability framework specified in D4.1.

Req 4-7 The UN must support metering of traffic at required levels.

The requirement 4-7 reports the need for mechanisms to ensure stability and responsiveness that could depend on the desired level of detail requested. From today's perspective these are at least at node level, at user/customer level and at VNF level.

Req 4-8 The UN must support mechanisms to ensure stability and responsiveness in any situation.

Requirement 4-8 details an important requirement from operational point of view. For example, even if a certain virtual network function running on the UN crashes, it must be possible to reach the management functions of the UN. The same principle applies for overload situation of traffic forwarding.

5 Design principles and general properties of the UNIFY architecture

5.1 Introduction

The previous sections outlined an overview of the technical setting, potential use cases and related requirements needed for identification of the design goals towards the UNIFY architecture. This section instead reports more in details the design principles on which the UNIFY architecture is going to be conceived.

Existing state of the art frameworks provide numerous meaningful ideas and design principles for the UNIFY architecture. Especially the principles of abstraction, layering and partitioning as well as flexibility in combining of different network functions and a process framework will be of huge benefit in designing the UNIFY solution. The most important aspects covered by this section are:

- Layering and recursion allowing separation of complexity
- Abstraction providing a way to separate semantics and implementation details
- Virtualisation allowing the logical separation of physical resources
- Processes describing the behaviour of the system
- Technology independence supporting generality of the approach (a major argument for the so-called "narrow-waist" design principle, see section 5.5 for details)
- Legacy support in terms of integration of existing modules and reuse of existing implementations (this supports the narrow-waist design principle)

So far the use case groups provide an idea about how and where to partition, layer or split the UNIFY architecture in order to reflect technical challenges, administrative borders, etc. According to this, first there is the separation between data and control layers with extension to install processing entities on demand. Second, abstraction and several layers are introduced in order to split between infrastructure, orchestration with management/control and service layer. In this scenario certain sub-layers could be identified as well and will be discussed later. Third partitioning and layering are needed in order to integrate other providers.

Starting from the use cases and state of the art analysis, 46 requirements have been identified, targeting the most important aspects of dynamic and flexible service management. All requirements will be considered in the latter part of the UNIFY architecture development, after setting the main frame.

Finally, beside technical challenges, it is required to find a design approach which provides agility and stability while at the same time supporting the UNIFY objectives on providing basic pillars for the other technical activities beside the demand for being future-proof. This will be discussed in more detail in the next section.

5.2 Architecture design steps

Currently, it is widely acknowledged that future design of networks requires more open, extendable infrastructures than the existing ones (see [20] for a summary). So within the UNIFY project this has been acknowledged and so the objective to create a reference architecture for being future-proof has been defined.

A future-proof architecture requires a similar design process, and for this reason the simplest approach has been to split the complexity into separate levels allowing for:

- Project wide understanding of basics and interactions with other developments
- Different levels of information depending on reader's understanding and desire
- A general overview on the architecture and its principles and parts
- An increasing level of detail about the functionality and implementation of the architecture
- Agility for developments with lower level changes not depending on higher level implementations

While doing so the design process is split into three architecture levels:

1. Overarching architecture highlighting the basic building blocks and general principles, documented in section 6
2. Functional architecture combining the different basic building blocks and principles with processes represented in functional elements, process descriptions and interfaces, documented in section 7
3. System architecture with further detailing of functional elements, integration of existing developments, selection of implementation, etc., documented in D2.2

5.3 Layering and recursiveness

There are several disadvantages known with layering, partitioning and splits in general. Typically, there is some state in each of the layers which needs to be synchronised in one or the other way. In addition, each layer adds headers or additional information to the packet that guides to additional problems like MTU size etc.

However, layering is widely accepted as a mechanism to separate problem solving accordingly to different concerns in order to achieve:

- Separation of concerns, information details, complexity and thereby providing scalability
- Possibility to (independently) revise smaller components
- Possibility to converge different implementations of one layer with the neighbouring layer
- Possibility to integrate existing solutions
- Possibility to hide implementation details
- Enabling split between different actors / roles

In the design of the UNIFY architecture most of these issues will appear and layering will be taken into account as a handle these issues. Especially, the following advantages could be realised:

- Scalability

    - In distributed computation for e.g. calculation of events in the control plane reducing the amount of events that propagate between control plane elements
    - Simplified topology view by abstracting nodes and links for forwarding state calculations,
    - Encapsulating failure monitoring and handling to smaller domains
- Simplifications in processes and methods by hiding/abstracting details
- Administrative partitioning for sub-domains

Recursiveness must be seen as an application of layering. Each layer is based on the same principle structure and performs the same tasks on a different scale. It can be also seen as an inherent property of virtualization and abstraction (details in section 5.4) on the control plane.

For example an SDN domain with three OpenFlow switches (A, B, C) and an out of band control network (red) is shown in the left part of Figure 5.1. Both the physical ports of the switches and the logical edge ports of the domain are shown in numbered circles. Assuming that OpenFlow "Switch C" is replaced with a domain consisting of OpenFlow switches "E" and "F" and a "Switch Agent C" mitigating a big switch control abstraction of the domain equivalent of a single switch, where C1, C2, C3 and C0 corresponds to 1, 2, 3 and 0 ports of C respectively (see Figure 5.1 middle part). From an SDN Controller or any other control application point of view, it is indistinguishable, whether a physical switch "C" or an equivalent domain abstraction is deployed in the network. However, "Switch Agent C" is an application running on top of an SDN controller for the corresponding domain (see Figure 5.1 right part). The controllers and control apps deal with different network abstractions according to their role and scope in the hierarchy. So it can be concluded that recursiveness is inherent to SDN's control plane abstraction and virtualization, therefore in UNIFY must support recursive hierarchy of domains.

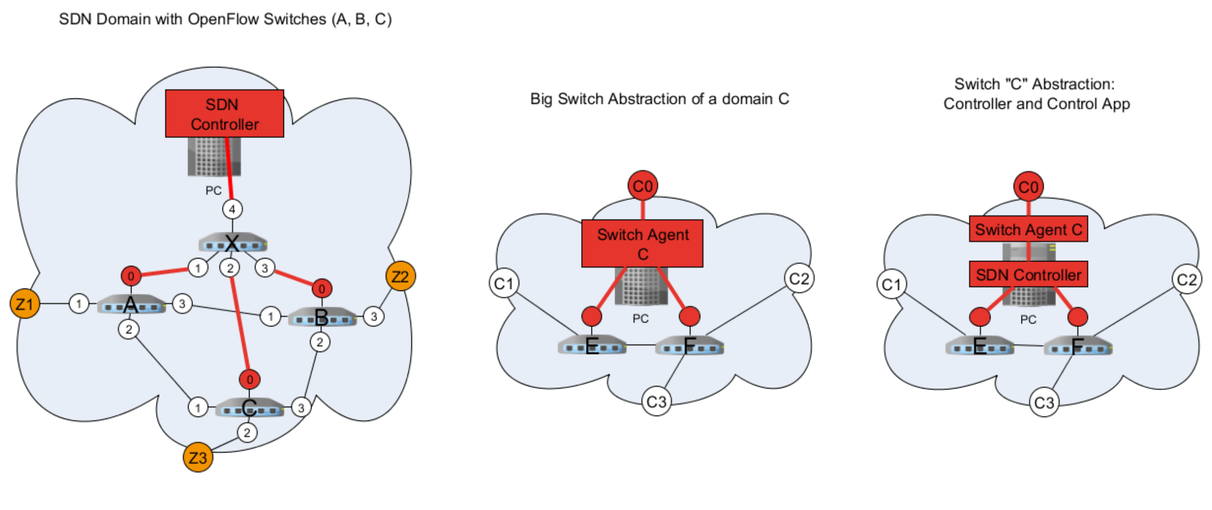

SDN Domain with OpenFlow Switches (A, B, C)
SDN Controller
PC
X
A
B
C
Z1
Z2
Z3
0
1
2
3
4
Big Switch Abstraction of a domain C
C0
Switch Agent C
PC
E
F
C1
C2
C3
Switch "C" Abstraction: Controller and Control App
C0
Switch Agent C
SDN Controller
PC
E
F
C1
C2
C3

Figure 5.1: SDN Domain with OpenFlow switches (A, B, C) and abstractions for Domain "C" and Switch "C"

5.4 Abstraction and virtualization

Abstraction[2] is a way of representing an entity or concept through a dedicated selection of potentially transformed characteristics. It enables the definition of functional components using a wide range of similar, but different types of entities. For example, while a network controller can be in control of many vendor-specific network nodes, using an abstracted view of these switches, it can handle them as similar, perform path-calculations on the resulting topology and steer the switches to provision such a path.

Adequate abstraction leads to clean functional architectures encouraging reusable components. In the words of Barbara Liskov: "Modularity based on abstraction is the way things get done", [21]. The core challenge of abstraction is the choice of selected characteristics. Whereas SDN has enabled a revolution in network abstraction by providing concepts such as open data plane interface (e.g., OpenFlow), UNIFY targets a harmonized, unified abstraction of both traditional cloud and telecommunication infrastructure based services.

In order to achieve this, UNIFY will focus on the abstraction and virtualization of CPU, storage and networking resources. The identified key abstractions of UNIFY in the context are:

- Service is considered to be representable as a chain (service function chain) or interconnection (service function graph) of (virtual) network functions.

Network and computing nodes are abstracted resources with ports, processing and/or forwarding capabilities and switching capacity in between. For example, all forwarding entities are abstracted as switches, servers are abstracted to a combination of cores with particular clock frequency, amount of RAM and HDD of given size and Universal Nodes are abstracted as a combination of network and storage/computing resources. In general, virtualization refers to the idea of a virtual instance of an (abstracted) resource rather than the actual resource itself. In many of the cases virtualization allows sharing the physical resource between different virtual instances at the same time. Such resource sharing requires management and control capabilities from the virtualization framework and some resource isolation capabilities if any guarantees are to be delivered.

The general concept of virtualization has been applied to different types of resources. For example, virtualised network functions can be implemented as a full VM using a server node virtualization based on hypervisor (e.g. QEMU, Xen, KVM, and VMware), as a Linux Container (e.g. LXC) or as a Click router process.

Networking resources can be also used to implement virtual network functions. The main challenge is proper isolation in the implementation. Network virtualization can be achieved by using an external entity (e.g. proxy-based solutions at control plane, such as FlowVisor and OpenVirteX) or at node level (e.g. Logical Switch Instances or Virtual Switch Instances). The mix of a proxy-based approach and a node-level approach can be also investigated to get the adequate level of virtualization (trade-off between flexibility and performance).

In the context of UNIFY, defining a common interface to manage all the computing and networking virtualization and the combination of both are the most relevant topics to be analysed. Especially, the mentioned joint interface for provisioning both types of resources and their actual interaction will be

[2] Freely adapted on ONF-based definition of abstraction and virtualization

addressed. The three different use case groups (section 3.1 - 3.3) involve different levels of virtualization (and thus abstraction as well). UCG1 focuses on virtualization of network functions (similar to ETSI ISG Network Functions Virtualization (NFV)). The idea behind NFV is the virtualization of the functions performed today by dedicated hardware appliances deployed by network operators in order to be run by general purpose compute resources (i.e. not specialized for such a function) instead. UCG2 addresses the concept of Service Function Chaining (SFC) (see section 2.7.1) where network and computing resources being virtual or physical are combined in order to interconnect those services functions and provide them in the optimal way. Last but not least, a multi-tenant scenario is explored in which different entities share the same physical resources to deliver services (see use case group 3, section 3.3). In this context, the virtualization of these resources is key to facilitate the provision of those services in a cost-effective manner and simplifies the management and control of such scenario.

To sum up, UNIFY combines both networking and cloud resources and focuses on providing processing capabilities to realize network functions for service delivery. The abstraction and virtualization of network functions is the basis to build a multi-tenant solution over the same infrastructure, and allows the dynamic adaptation of resources to the actual demand. UNIFY explores how to provide those virtual network functions using any types of resources like general x86 processing platforms and advances beyond legacy IT virtualization techniques based on computing resources with an integrated, unified computing and network provisioning and operation concept.

5.5 Narrow waist

One of the most well-known designs following the narrow waist principle is the IP [22]. IP defines the bare minimum of functionality (logical addressing, fragmentation and reassembly and connectionless packet forwarding) in its interoperation layer (Layer 3 or IP), while brings the elegance of design diversity into above and below. Even though such a narrow waist design is extremely difficult to change in its minimal functions, if properly selected it allows unlimited innovation around. For example, IP hourglass completed with TCP and UDP allowed innovation in the desktops and servers for unlimited number of applications that exist today. Therefore, even researchers proposing novel replacements of IP widely assume a narrow waist point in their architectures.

Also, narrow waists emerged in other areas. The hardware virtualization (virtual machine abstraction) for cloud environment created its own narrow waist in low level compute and storage virtualization. Virtual Machine (VM) abstractions are compatible with legacy operating systems and their valuable ecosystem of applications.

In the SDN domain, seeking for the new narrow waist is ongoing. OpenFlow could be considered as one option here with many alternatives. Dan Pitt (ONF Chair) in [23] pointed out that standardization of such single narrow waist interface is still premature in the SDN space considering the many alternatives. Neela Jacques of the OpenDaylight project envisioned that there might be multiple narrow waists emerging in different parts of the ecosystem [23].

All in all, the community seeks for a narrow waist to combine networking and cloud with the benefits of open innovation and the support of legacies. In UNIFY we seek for a narrow waist control plane definition, which creates a harmonized compute and networking abstraction and virtualization for generic control needs. This would allow defining processing anywhere in telco service offerings.

5.6 Definition of processes

As briefly discussed in section 2.9 and section 2.10, a process oriented analysis and design is an important aspect in the definition of the architecture. Therefore a process model needs to be defined and in the definition of the UNIFY architecture taken into account. This process model will be based on the TMForum Business Process Framework (eTOM) and its process model as well as the basic commonalities with the DevOps concept although limited to the basics of the relevant processes only. Basics of eTOM are identified and documented in Annex 5. In addition, an analysis of the impact of the basic process areas and most important aspects is included in this annex. In the following, the guidance is for process definition is based on this analysis and it is not claimed to fully represent eTOM or to model all processes in detail. A more fine granular level (so called level 2) is depicted in Figure 5.2 below.

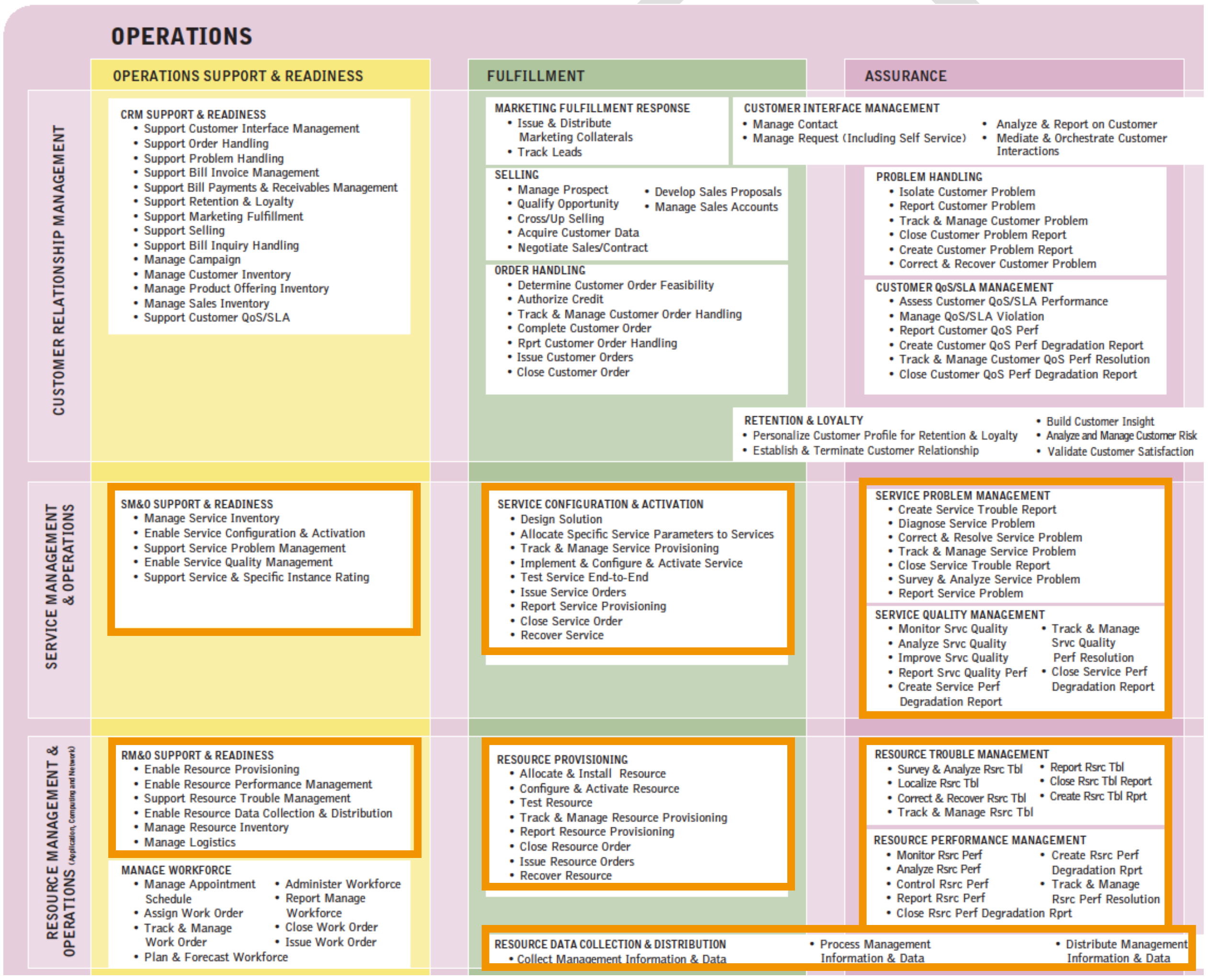

OPERATIONS
OPERATIONS SUPPORT & READINESS
FULFILLMENT
ASSURANCE
CUSTOMER RELATIONSHIP MANAGEMENT
SERVICE MANAGEMENT & OPERATIONS
RESOURCE MANAGEMENT & OPERATIONS (Application, Computing and Network)
CRM SUPPORT & READINESS
• Support Customer Interface Management
• Support Order Handling
• Support Problem Handling
• Support Bill Invoice Management
• Support Bill Payments & Receivables Management
• Support Retention & Loyalty
• Support Marketing Fulfillment
• Support Selling
• Support Bill Inquiry Handling
• Manage Campaign
• Manage Customer Inventory
• Manage Product Offering Inventory
• Manage Sales Inventory
• Support Customer QoS/SLA
MARKETING FULFILLMENT RESPONSE
• Issue & Distribute Marketing Collaterals
• Track Leads
CUSTOMER INTERFACE MANAGEMENT
• Manage Contact
• Manage Request (Including Self Service)
• Analyze & Report on Customer
• Mediate & Orchestrate Customer Interactions
SELLING
• Manage Prospect
• Qualify Opportunity
• Cross/Up Selling
• Acquire Customer Data
• Negotiate Sales/Contract
• Develop Sales Proposals
• Manage Sales Accounts
ORDER HANDLING
• Determine Customer Order Feasibility
• Authorize Credit
• Track & Manage Customer Order Handling
• Complete Customer Order
• Rprt Customer Order Handling
• Issue Customer Orders
• Close Customer Order
PROBLEM HANDLING
• Isolate Customer Problem
• Report Customer Problem
• Track & Manage Customer Problem
• Close Customer Problem Report
• Create Customer Problem Report
• Correct & Recover Customer Problem
CUSTOMER QoS/SLA MANAGEMENT
• Assess Customer QoS/SLA Performance
• Manage QoS/SLA Violation
• Report Customer QoS Perf
• Create Customer QoS Perf Degradation Report
• Track & Manage Customer QoS Perf Resolution
• Close Customer QoS Perf Degradation Report
RETENTION & LOYALTY
• Personalize Customer Profile for Retention & Loyalty
• Establish & Terminate Customer Relationship
• Build Customer Insight
• Analyze and Manage Customer Risk
• Validate Customer Satisfaction
SM&O SUPPORT & READINESS
• Manage Service Inventory
• Enable Service Configuration & Activation
• Support Service Problem Management
• Enable Service Quality Management
• Support Service & Specific Instance Rating
SERVICE CONFIGURATION & ACTIVATION
• Design Solution
• Allocate Specific Service Parameters to Services
• Track & Manage Service Provisioning
• Implement & Configure & Activate Service
• Test Service End-to-End
• Issue Service Orders
• Report Service Provisioning
• Close Service Order
• Recover Service
SERVICE PROBLEM MANAGEMENT
• Create Service Trouble Report
• Diagnose Service Problem
• Correct & Resolve Service Problem
• Track & Manage Service Problem
• Close Service Trouble Report
• Survey & Analyze Service Problem
• Report Service Problem
SERVICE QUALITY MANAGEMENT
• Monitor Srvc Quality
• Analyze Srvc Quality
• Improve Srvc Quality
• Report Srvc Quality Perf
• Create Service Perf Degradation Report
• Track & Manage Srvc Quality Perf Resolution
• Close Service Perf Degradation Report
RM&O SUPPORT & READINESS
• Enable Resource Provisioning
• Enable Resource Performance Management
• Support Resource Trouble Management
• Enable Resource Data Collection & Distribution
• Manage Resource Inventory
• Manage Logistics
MANAGE WORKFORCE
• Manage Appointment Schedule
• Assign Work Order
• Track & Manage Work Order
• Plan & Forecast Workforce
• Administer Workforce
• Report Manage Workforce
• Close Work Order
• Issue Work Order
RESOURCE PROVISIONING
• Allocate & Install Resource
• Configure & Activate Resource
• Test Resource
• Track & Manage Resource Provisioning
• Report Resource Provisioning
• Close Resource Order
• Issue Resource Orders
• Recover Resource
RESOURCE TROUBLE MANAGEMENT
• Survey & Analyze Rsrc Tbl
• Localize Rsrc Tbl
• Correct & Recover Rsrc Tbl
• Track & Manage Rsrc Tbl
• Report Rsrc Tbl
• Close Rsrc Tbl Report
• Create Rsrc Tbl Rprt
RESOURCE PERFORMANCE MANAGEMENT
• Monitor Rsrc Perf
• Analyze Rsrc Perf
• Control Rsrc Perf
• Report Rsrc Perf
• Close Rsrc Perf Degradation Rprt
• Create Rsrc Perf Degradation Rprt
• Track & Manage Rsrc Perf Resolution
RESOURCE DATA COLLECTION & DISTRIBUTION
• Collect Management Information & Data
• Process Management Information & Data
• Distribute Management Information & Data

Figure 5.2: Relevant UNIFY processes at eTOM level 2

Going into the details of Figure 5.2, the following aspects are of importance for UNIFY:

- Service Management and Operations with all on service inventories, base configurations for service deployment, service configurations and activations as well as related problem and quality management
- Resource management and operations support for resource provisioning enablement, base configuration and inventory services
- Resource provisioning with processes related to dedicated fulfilment for resources
- Resource trouble and performance management
- Resource data collection and distribution

Some aspects of these processes are out of scope. First of all, the process on managing work force while being a management process which does require technical assistance. Second, any process dealing with logistics is out of scope, it is assumed that the hardware is at least available to the framework. Third, there should be no principal logical and practical split between services and resources in a layered architecture: from a higher layer, lower layer offerings can be simplified to "resources" and from the lower layer, everything offered to a higher layer can be simplified to "services".

Important for the overarching and the functional architecture are the following logical basic building blocks:

- Inventories storing all required information
- Interfaces and functions allowing other operational processes (e.g. configuration or monitoring activities) to connect to service and resource entities
- Functions and interfaces allowing the configuration of resources for specific services
- Functions understanding errors, failures and quality levels
- Functions and interfaces collecting information about resources and services

In the process to define the architecture in more detail, especially on the level of the system architecture, the list of fine-grained, defined processes of Figure 5.2 will be detailed.

Besides identifying the processes that need to be performed within a lifecycle of service graphs (the WHAT), it is important to note that eTOM here has primarily been used to help in. However, the implementation of these processes is left out to some extent (the HOW).

As an example, UNIFY has to support operator teams performing the development, fulfilment and assurance processes with access to the same tools and methods. As a result the selected processes in the model need to be enhanced with technology support by the DevOps principles. In detail, DevOps principles are reflecting:

- Develop and test against production-like systems: With respect to development of new or updated virtual network functions (VNFs), we will detail a process supporting VNF development, allowing the developers of new or updated VNFs to deploy and verify their functions on the UNIFY production environment. With respect to development (or rather definition) of service graphs, the developers are supported by verification and debugging capabilities within the various layers of the architecture. And with respect to the operation of the infrastructure, isolation and virtualisation will be key concepts to protect other active services

- <u>Deploy with repeatable, reliable processes</u>*:* This principle corresponds largely to one of the UNIFY core ideas about automatic deployment of service chains. Reliability of the process is supported by observability and verification capabilities available for each step of the S*ervice Deployment* process
- <u>Monitor and validate operational quality</u>*:* this principle is reflected by the assurance process before
- Amplify feedback loops: This principle is very much connected to both the culture within an organization, as well as business decisions based on market or customer feedback. In UNIFY, this principle is obeyed from a technical point of view by providing a customer facing service layer, including interfaces towards fulfilment and assurance processes, potentially even development processes

6 Overarching Architecture

6.1 Introduction

The overarching architecture is the first design step for the UNIFY architecture. It is required for combining the design principles with each other while taking into account the use cases and their technical environments as well as reflecting the requirements. In addition, an alignment of the technical work packages on Orchestration and Programmability, Service Provider DevOps and on the Universal Node is required for the final combination of the individual developments in an integrated prototype.

As reported in the previous sections of this document, the three step design approach was selected with sequential refinement of the level of details (see section 5.2), starting from an overarching architecture, followed by the functional architecture (see section 7) and finally detailing the system architecture (will be documented in D2.2).

This section documents the resulting overarching architecture, which combines layering, abstraction, virtualisation, recursion and processes with the required functionality for Orchestration and Programmability, the Service Provider DevOps and the Universal Node. The architecture is intended to apply on telecommunication networks in general, including corner cases such as data centres and home network environments.

This chapter is structured as follows. First, the general architecture layers and their required interfaces are defined. Second, an introduction to recursion in the architecture with examples is given. Third, the application of the defined processes is explained. Fourth, a special section is dedicated to management. And finally, the last two sections deal with the potential business roles in the defined architecture and with the mapping to well-known industry initiatives ETSI NFV and ONF.

6.2 Architecture layers and interfaces

As stated before (section 5.3), layering is one of the fundamental design principle. The main question is how and where to layer, which will be answered by analogies and conclusions from the use cases and requirements.

First of all, there are two main layers introduced, application and infrastructure. The first one entails the user and his applications, the latter one summarizes the existing resources (for example compute, storage and networking) as infrastructure. Second, there must be at least one layer that connects these two layers and allows flexibility, extensibility and openness (see requirements in section 4.1).

Looking into best practises today, one could find two examples illustrating the general solution:

- SDN assumes a network operating system (NetOS), which orchestrates and abstracts forwarding resources and interfaces for control programs (or applications) to the (virtualised) resources (see section 2.1)
- Operating systems on standard computing hardware share resources between applications and provide possibilities to integrate the underlying resources in a plug and play manner (for instance adding USB drives)

Both examples are analogies for bridging between resources and applications in networks and single nodes: an operating system which among other functions provides integration and some orchestration, provisioning interfaces and control to resources. In the following, this layer is called Orchestration Layer.

The Orchestration Layer has the following responsibilities:

- Interacting with the Service Layer accepting Network Function Forwarding Graphs, sending reports and status information
- Providing harmonized and unified representation of all underlying physical and virtual resources and their capabilities
- Providing common enablers for service (function) chaining, defined as logical interconnection of network functions
- Handling of resource requests
  - Analysis of Network Function Forwarding Graphs, identification of endpoints and required capabilities, understanding of quality indicators
  - Mapping of resources, capabilities and quality indicators in order to satisfy Network Function Forwarding Graphs, potentially split into small chunks of Network Function Forwarding Graphs
  - Requesting reservation and configuration of the Network Function Forwarding Graphs including monitoring, trouble-shooting functions and forwarding in-between domains
  - Translating the request to different local orchestrators interface properties (if required)
  - Awaiting approval and successful reservation
  - Instantiation and configuration of resource reprovisioning interfaces for services
- Providing feedback information about the state of resources
- Automated, plug and play management of components and elements
- Support of existing implementations, like interfaces to existing controller frameworks

As a consequence, the following points relative to the interfaces need to be considered:

- Interfaces to the Service Layer operate on generic resources and capabilities, i.e., generalized resources descriptions corresponding to virtualized network, compute and storage resources. These interfaces also provide monitoring and steering information. One potential design goal for the resources representation is a narrow-waist (see section 5.5 for discussion).
- The Orchestration Layer focuses on dynamic allocation of resources with requirements that change over time; no service function logic is included in the Orchestration Layer

- Northbound interface should be technology independent; in contrast the southbound interface will support multitude of interfaces which are potentially technology dependent
- Southbound interfaces have to take layering and recursion into account
- Southbound protocols need to be adapted in order to support the resource control mechanisms for handling resource requests, state information of resources, etc.
- Supporting interfaces for monitoring and transmission of status and notification of computation, storage and network resources

The resulting layers and components with the interconnecting interfaces are shown in Figure 6.1.

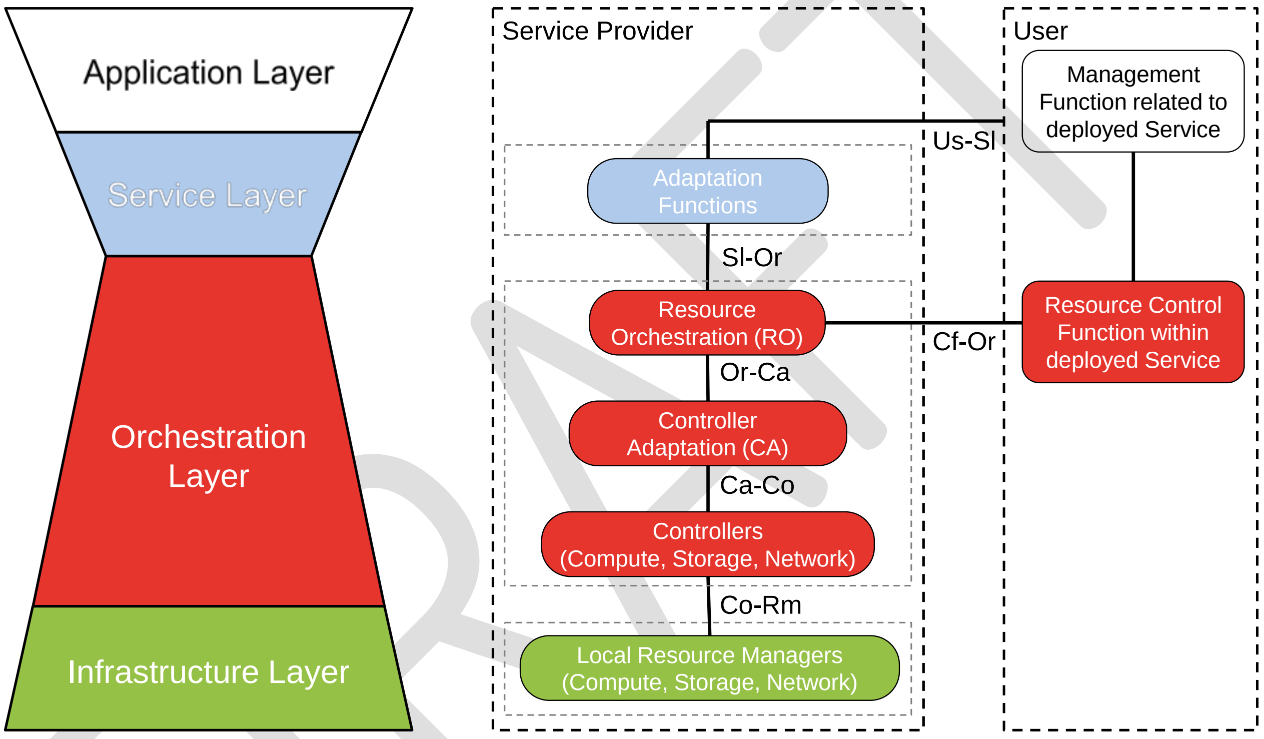

Application Layer
Service Layer
Orchestration Layer
Infrastructure Layer
Service Provider
User
Management Function related to deployed Service
Us-Sl
Adaptation Functions
Sl-Or
Resource Orchestration (RO)
Cf-Or
Resource Control Function within deployed Service
Or-Ca
Controller Adaptation (CA)
Ca-Co
Controllers (Compute, Storage, Network)
Co-Rm
Local Resource Managers (Compute, Storage, Network)

Figure 6.1: The three layered UNIFY architecture

The Service Layer (blue in Figure 6.1) turns the service chain provisioning into consumable services by defining and managing service logics; establishing programmability interfaces to users (residential, enterprise, network-network, OTT, etc.) and interacting with management functions associated to deployed services and traditional OSS/BSS systems. The Service Layer is also responsible to create further service abstractions as needed toward the different users (e.g., providing a BigSwitch topology that abstract details of the physical topology) and to realize the necessary adaptations according to such abstractions.

The Orchestration Layer (red in Figure 6.1) is split into three major sub-components: resource orchestration, controller adaptation, and controllers. The resource orchestration is a logically centralized function. Below, there could be many underlying controllers corresponding to different domains or technologies in practice. The controller adaptation is responsible to bridge between the controllers and resource orchestrators. It offers technology independent, virtualized resources and resource information; and translates resource requests into commands appropriate for a particular controller implementation. Hence, the resource orchestration collects and harmonizes virtualized resources and resource information into a global virtualized resource view at its compute, storage and networking abstraction. It is important to note here, that the aim of the resource orchestration is to collect global resource view and this is available to the Service Layer intact.

The global resource view in the orchestrator consists of four main components; forwarding elements, compute host capabilities, hardware based or accelerated network function capabilities, and the data plane links that connect them. All of the resources must have associated abstract attributes (capabilities) for the resource provisioning to work. The additional functionality needed in the Orchestration Layer to be able to map services and service chains (re-)provisioning request to these global resources will be developed in the orchestration and programmability framework and documented in D3.1 and D3.2.

The Infrastructure Layer (green in Figure 6.1) encompasses all networking, compute and storage resources. By exploiting suitable virtualization technologies this layer supports the creation of virtual instances (networking, compute and storage) out of the physical resources. Primarily, four domains of physical resources are considered:

- Universal Node (see D5.2 for details)
- SDN enabled network nodes (like OpenFlow switches)
- Data Centres (like controlled by OpenStack)
- legacy network nodes or network appliances, where at least a logically central entity provides abstraction services for the legacy domain

Figure 6.1 illustrates a number of interfaces between the different components:

- Us-Sl provides user – system interaction in order to express services and translate them into Service Graphs (SG). In practice, this could be a web interface.
- Sl-Or provides information and management actions to the translated service graph as Network Function Forwarding Graph (NF-FG).
- Or-Ca provides an interface for passing a complete or partial Network Function Forwarding Graph (NF-FG) which needs translation into controller specific commands and resource specific information
- Ca-Co provides translation service for information and management actions to a complete or partial Network Function Forwarding Graph (NF-FG)
- Co-Rm provides information and management actions to the local agent who controls the resources in the Infrastructure Layer
- Cf-Or provides information and management actions to the resource control function in a deployed service for (re-)provisioning tasks

Section 6.4 provides further details for the programmability related interfaces Sl-Or, Or-Ca, Or-Ca, Ca-Co and Co-Rm and the associated management actions. A list of potential users can be found in section 6.6.

6.3 Recursiveness, virtualisation and abstraction

In the sections 5.3 and 5.4, the fundamentals on recursiveness, virtualisation and abstraction have been presented. One important aspect was that recursiveness is an inherent property of virtualisation and abstraction in the control plane. This for example means that a higher layer cannot verify whether it is accessing abstracted resources or not in the lower layer unless it is notified or additional mechanisms are trying to verify it.

In the following, two examples illustrate the most probable recursions in the UNIFY architecture. Please note that details require further refinement and will be documented in D2.2.

The first example is the Universal Node (UN) and the integration with global orchestrator combining different UNs shown in Figure 6.2. From a logical point of view the UN consists of the Orchestration Layer and Infrastructure Layer. The Orchestration Layer (named as Unified Resource Manager) is responsible for collecting the local resources and adaptation inside the node and providing it to the overarching, global orchestrator. In practice, the local orchestrator could be simplified as it does not need to support southbound in the same way as a global entity has to. More details can be found in D5.2 on the Unified Resource Manager, VNF Execution Environment and VSE modules of the UN. The recursion is applied with the northbound interface of the local orchestrator in the Unified Resource Manager (Ca-Co instantiated as Sl-Or). Here the interface to the global orchestrator is the same as the global orchestrator's northbound interface to the Service Layer with less resource information (Sl-Or). The other direction is similar. The global orchestrator's resource orchestration receives service requests from the Service layer and is able to delegate them to lower layer orchestrator (local orchestrators) using a similar interface (Sl-Or) as the one between Service Layer and Orchestration Layer. The interface of the Orchestration Layer in the Unified Resource Manager of the UN follows the same structure. Both interfaces (Sl-Or and Sl-Or instantiation of Ca-Co) could be differently implemented but have to support the same functionality.

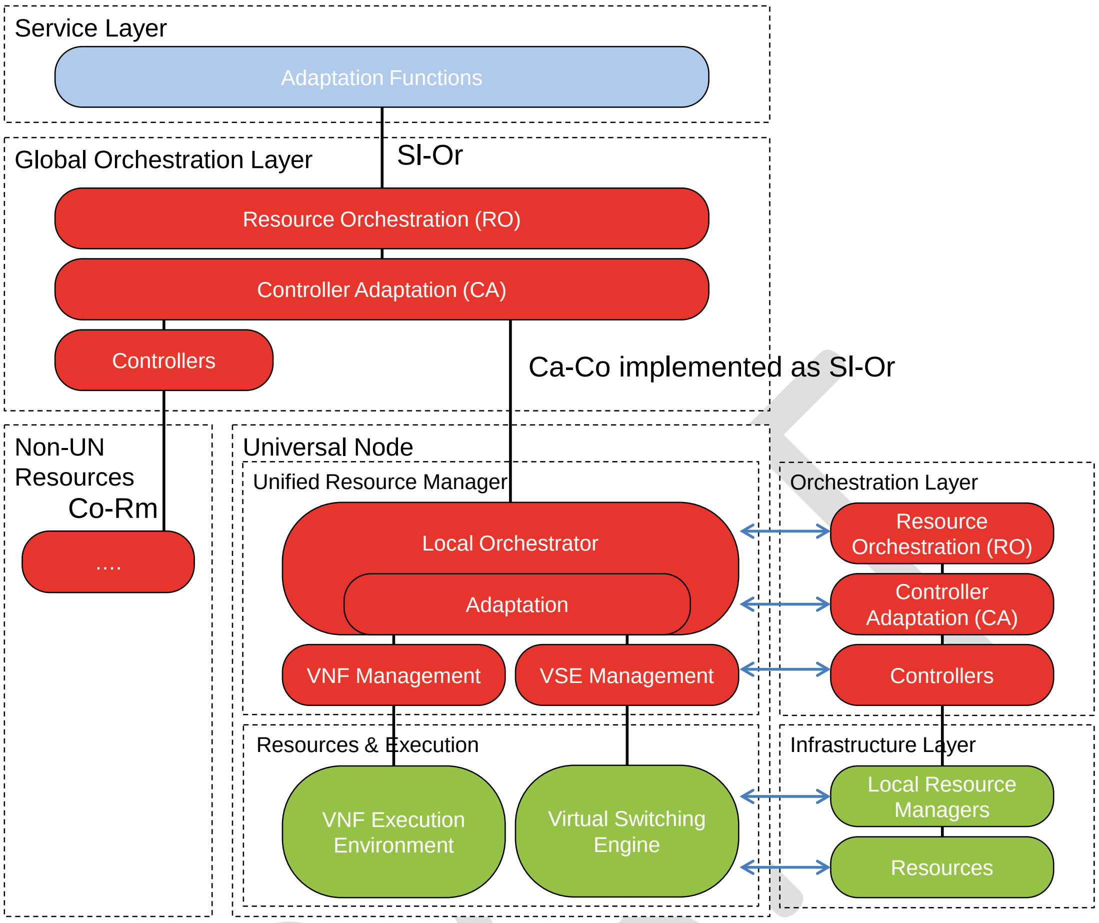


Figure 6.2: Example on recursion in UNIFY architecture with Universal Node

The second example is about splitting a Network Function Forwarding Graph across different administrative domains as shown in Figure 6.3. This could be required for different reasons, e.g. different business entities, logical structure of the infrastructure, etc. Again, the global orchestrator interconnects different local orchestrators and offers to a Service Layer the complete resource view. From a functional point of view, the different local orchestrators could represent a single node, a group of nodes of the same type, a group of nodes of different types or other orchestrators. These local orchestrators provide an aggregated resource view to the global orchestrator, which in turn acts and is seen from the local orchestrator as service layer.

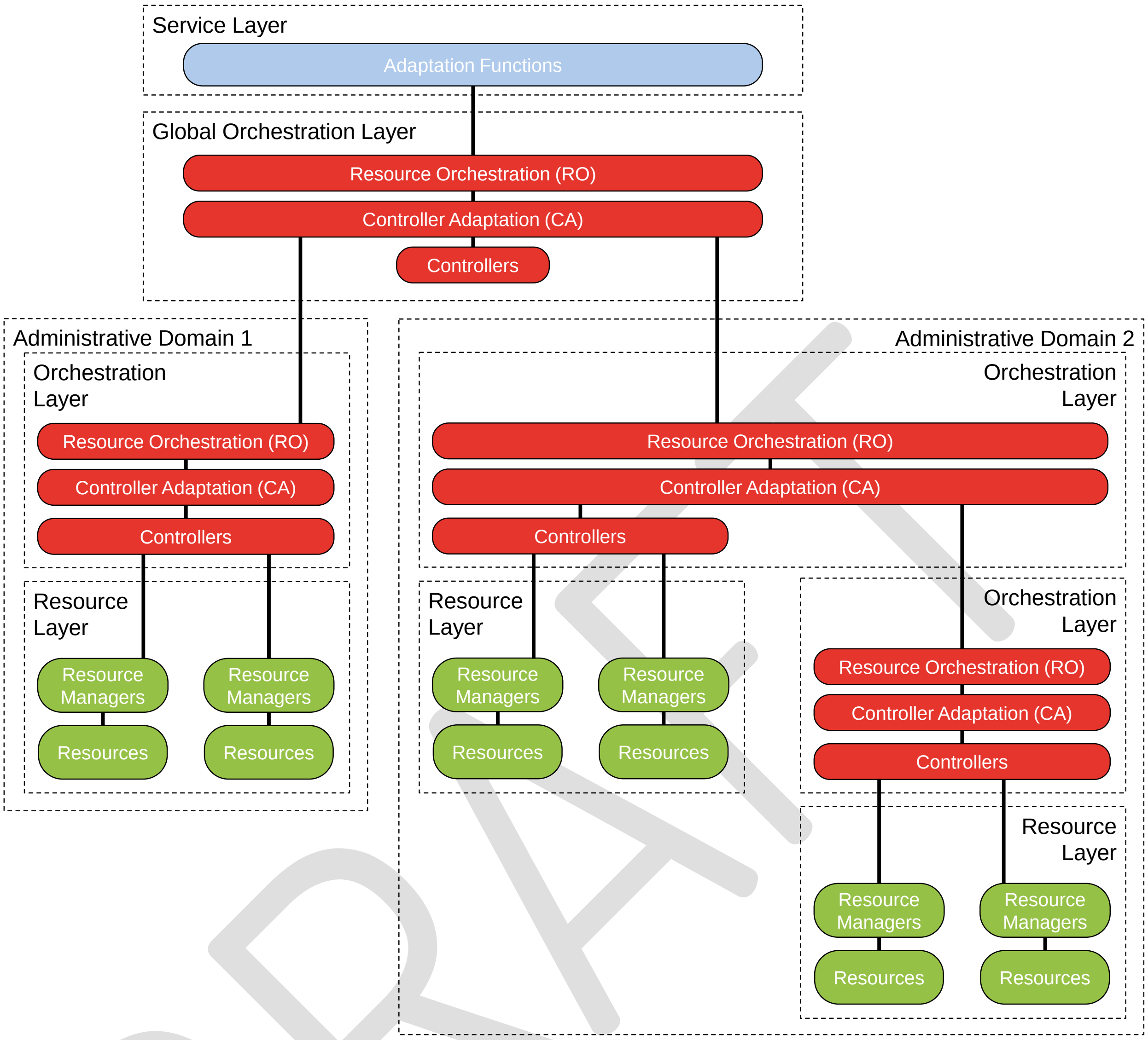


Figure 6.3: Example on recursion in UNIFY architecture with multiple domains

More in detail, the global orchestrator has the following adopted tasks compared to section 6.2:

- Map the Network Function Forwarding Graphs to the different local orchestrators
- Request execution of the resource requests at the local orchestrators
- Collect resource state information from the local orchestrators, analysing resource performance and service quality indicators

The local orchestrators have similar functionality within their domain. Recursion is represented in this example by the interaction in-between Service Layer, global orchestrator and local orchestrator (using the Ca-Co interface implemented as Sl-Or interface), which all act via the same interface properties:

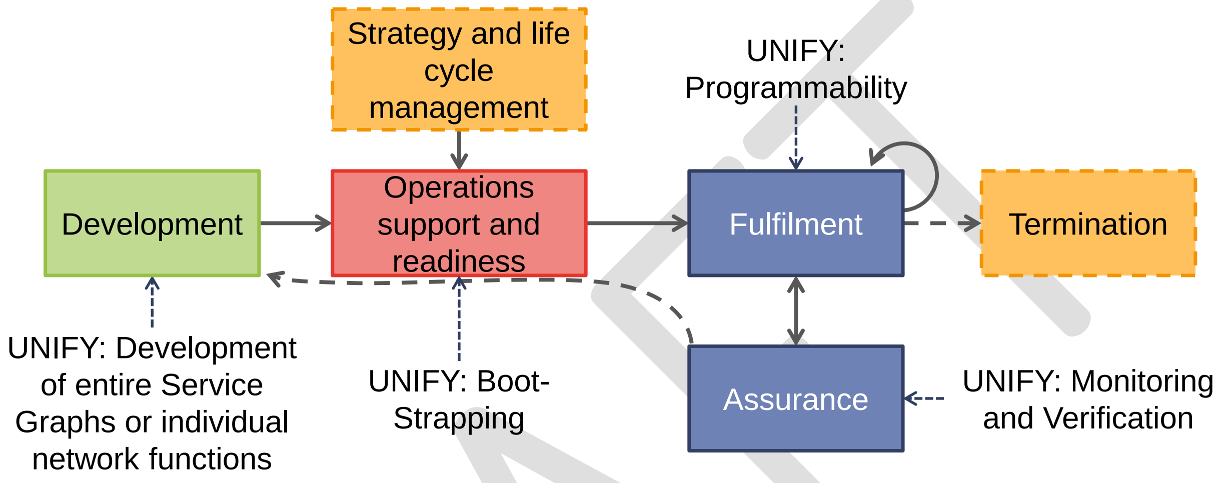
- Service Layer acts as a controlling entity for the global orchestrator which represents a data path entity
- Similarly, the global and local orchestrators represent a controlling entity and its data path entity

The principles of the three layers and recursion have been shown in these two examples already. In addition, an introduction to the functionality of the orchestrator and the programming of services is given. More details for this and additional three processes are given in the following section.

6.4 Detailing of defined processes

6.4.1 Introduction

The definition of processes in section 5.6 is summarized with the development of a simplified process model (Figure 6.4). In the following the main processes are captured in detail.



Strategy and life cycle management
UNIFY: Programmability
Development
Operations support and readiness
Fulfilment
Termination
Assurance
UNIFY: Development of entire Service Graphs or individual network functions
UNIFY: Boot-Strapping
UNIFY: Monitoring and Verification

Figure 6.4: Simplified Process model and mapping to UNIFY

UNIFY will have four main process parts:

- Boot-strapping is covering all aspects to enable the whole system environment to work. This includes details to collect information in database, to inform the logical elements on the different layers about available information and to enable infrastructure to perform basic management tasks
- Programmability is covering all process aspects of analysis, as well as configuration of services and sub sequentially resources. Programmability will reside on available information and interfaces from the boot-strapping process
- Monitoring and verification is covering all operational aspects like collection of information, verification, analytics, failure detection and resolution, quality observations. The process will reside on information from boot-strapping process, as well as interaction with programmability for monitoring function placement and configuration
- The Development process is covering the capture of the aspects of dynamic development of network functions on the one hand as well as the development (or rather definition) of service graphs on the other hand.

Orchestration cannot be directly mapped to the process model; it consists of programmability and parts of operation support and readiness as well as assurance.

Moreover, UNIFY has two additional processes:

- A termination process is included for completeness. It provides a symbolic placeholder and will be left out in the initial detailed analysis. Its impact on the design will be further studied in subsequent work items. Still it is assumed that it will have a minor impact on the general architecture and is therefore marked in a dotted line.
- Strategy and life cycle management is crucial in the sense to understand the general requirements of the business area, customer demands and resulting application and services. There will be no dedicated technical development in UNIFY (therefore marked in a dotted line), but it has an influence on many design aspects

In the following, each of the processes is described in detail.

6.4.2 Boot-strapping process

The purpose of the boot-strapping process is the introduction of basic management features to introduce automation inside the UNIFY production environment. It covers the operation support and readiness process as well as parts of the lifecycle management. This process has to ensure that:

- All physical nodes, management and control logic elements at and inside an architectural layer are configured and manageable for acceptance of configuration tasks, retrieval of monitoring information, etc.
- All physical nodes, management and control elements in the architecture are configured with connectivity information and are able to securely connect to configuring elements, e.g. controllers connected to orchestrators awaiting service requests, application control connected to Universal Node elements hosting the application execution environment
- Capability information is available at all appropriate levels and components of the architecture
- Abstractions and virtualization is understood at the different architectural layers and required transformations of e.g. address spaces in between different layers could be carried out

A number of the above mentioned tasks are already performed today, but with pre- or manual configuration and intervention which would hinder automation a lot. In addition, some task would require information management in data bases, something out of scope for UNIFY. Therefore, this section will concentrate on two things. First the configuration and integration of resources into the Unified production environment and second the required information exchange between Infrastructure and Orchestration Layer, which according to section 6.2 and the second example in section 6.3 could be connected with the global and local orchestrators as well. In the following, a number of things are indeterminate and therefore assumed to be available as generalised information, e.g. a template describing the capabilities of a node or virtual element, which from today's perspective is not necessarily given. Some aspects will be developed with the programmability framework and Universal Node concept, not all aspects will remain open and even though they are mentioned for completeness only.

The first description is the attachment of a physical node to the architecture, the configuration of connectivity information and the general announcement of its capabilities inside the architecture. A generalized process is described in Figure 6.5. The following elements are needed:

- Node representing a physical element, e.g. a server, switch or router
- Network configuration entity which if required provides basic network configuration service to nodes, e.g. a service running on a network controller
- Operations Manager responsible for directing all information in a domain and organising execution of service request in form of Network Function Forwarding Graphs
- Templates hosted in the Resource Database describing the capabilities of a node
- Configurations available in the Resource Database describing configuration tasks for nodes

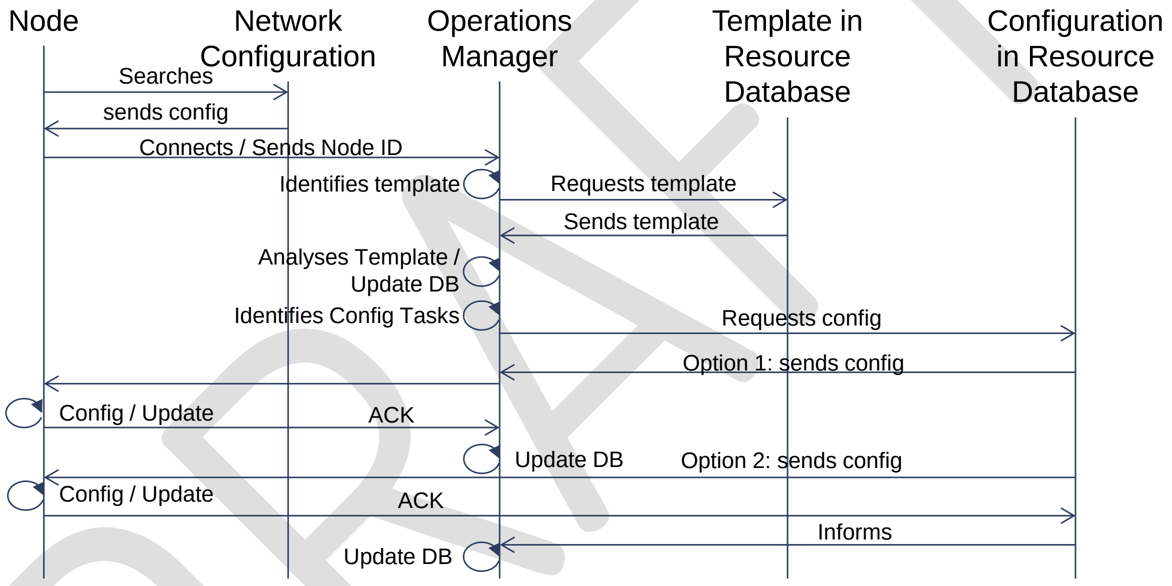


Figure 6.5: Boot-strapping process for connecting an element

The assumptions are that the node and its ID (for instance a MAC address of the management interface) are identified by the Operations Manager, the node template is available in the Resource database and the node ID is associated to the template, and the configuration tasks are described accordingly.

The process starts with an initial connectivity configuration of the node, so that it can find the central intelligence agent, namely the Operations Manager. This logical element knows all information for management and operation of the different resources, databases holding the information and elements which could perform dedicated control and management action. The central idea is that the node ID is used to identify the node in the Operations Manager, the appropriate template is selected among the Templates in the Resource Database and additional configuration tasks are analysed by the help of Configurations in the Resource Database. The NodeID must be unique and support a secure AAA process to prevent rogue insertion of Universal Nodes or any other element in the architecture. For the latter part of the process, two options are shown in Figure 6.5, both resulting in state information about the successful configuration and therefore availability of the node to the unified production environment. Potential configuration tasks are:

- Connectivity information, e.g. IP prefixes
- Orchestrator interfaces and potentially available/allowed orchestrators
- Device specific monitoring interfaces

It should be noted that certain parts of the process could be performed with other, automated mechanisms like zero-conf actions.

The second described process is the connection between two layers, essentially by exchange of resource information and required execution of configuration tasks. The process is required for attaching the Infrastructure Layer with the Orchestration Layer, for example a data centre represented by a single northbound interface of an orchestrator/controller or complete other administrative domains represented by an orchestrator's northbound interface. In both cases the representative domain is the Infrastructure Layer.

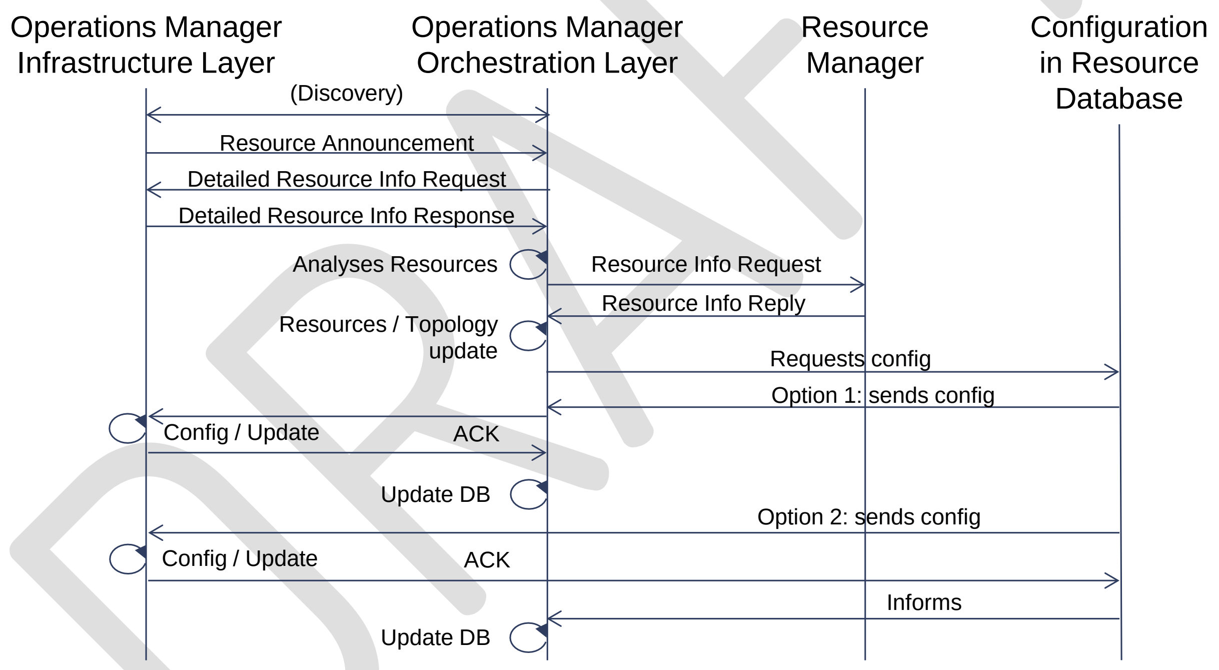


Figure 6.6: Boot-strapping process for connecting different layers

Again some assumptions for the process are made. First the connectivity of the two layers is available; authentication information for the layers is available and could be verified. The process starts with a discovery mechanism that finds the Operation Managers of both layers. Here information about the interface and authentication, the desired responsibilities, etc. is exchanged first. Second the resource information exchange starts. So the detailed resource information including e.g. information about special hardware is first announced, requested and finally transferred. At that point in time the Orchestration Layer analyses the resources, for example starts a topology discovery or uses neighbour information from the controllers, and stores the information in the Resource Manager. If required, the Configurations in Resource Database are consulted in order to perform configuration tasks in the Operations Manager or the infrastructure like continuity check of the operational state of the two domains/layers, access list configuration, encapsulation configuration for domain separation, etc.

6.4.3 Programmability process

The goal with the introduction of UNIFY's programmability framework is to enable on-demand processing anywhere in the physically distributed network and clouds. The major objective is to enable dynamic and fine granular service (re-)provisioning, which can hide significant parts of the resource management complexity from service providers and users, hence allowing them to focus on service and application innovation similarly to other successful models like the IP narrow waist.

The objective of the programmability process flow is to resolve and map network function forwarding graph (NF-FG) and associated requirements through different levels of abstractions to physical resources available in the distributed system (both network and cloud) while adhering to operational policies. In the following, the programmability process flow is described by reference points and corresponding information models. Further detailing will be available in D2.2.

6.4.3.1 Programmability Reference Points

For a top-down programmability flow, the service adaptation, the orchestration, the controller adaptation, controllers and local resource managers were identified as key components. The corresponding reference points are (see Figure 6.1):

- Us-Sl: Users (services) and the Service Layer.
- Sl-Or: Service Layer's adaptation function and the Resource Orchestrator
- Or-Ca: Southbound interface of the orchestrator toward an adaptation logic scoping and interfacing with various controllers.
- Ca-Co: Northbound interfaces of controllers.
- Co-Rm: Southbound interfaces of controllers.

One focus of UNIFY is to design Us-Sl, Sl-Or and Or-Ca reference points, while interfacing to various state of the art controllers available or under development through Ca-Co interfaces. The Sl-Or and the Or-Ca interfaces both use the same NF-FG description format, which used between different orchestrators as well (see example two in Figure 6.3). The Universal Node's northbound interface is also the same to the inter-orchestrator interface (Sl-Or).

6.4.3.2 Us-Sl reference point

A Service Graph (example provided in Figure 6.7) describes the service requested by a user and defines how and where the service is provided; and how successful delivery of the service is measured. The provided service is described by network functions /applications and their logical connectivity, and where the service is provided is represented by connectivity to Service Attachment Points. Finally Key Quality Indicators (KQIs) attached to both network functions and the logical connectivity describe how delivery is measured.

The network functions defining the service at this level may be either Elemental Network Functions (ENF) which perform a specific function (such as a NAT, Traffic classifier, transparent HTTP Proxy, Firewall function, etc.) or a Compound Network Functions (CNF) that internally consists of multiple ENFs. A CNF performing for example a parental control function could internally consist of a sub-graph starting with a traffic classifier followed by a HTTP proxy and firewall. When traffic passes through the classifier it could inform the following functions and steer traffic to either of them for either re-writing parts of the HTTP requests/replies (for example certain image URLs) in the HTTP proxy or fully blocks the flow in the firewall. In the Service Graph we make no distinction whether the requested function is a CNF or ENF, they are both represented simply as network functions / applications. CNFs will be resolved by lower layer functions, see next section for details.

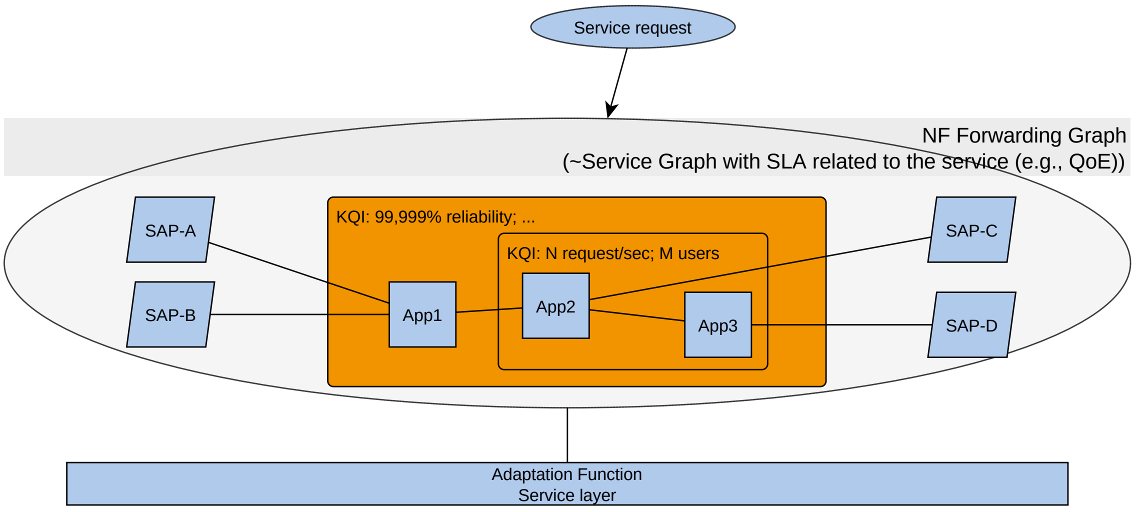

Service request
NF Forwarding Graph
(~Service Graph with SLA related to the service (e.g., QoE))
SAP-A
SAP-B
KQI: 99,999% reliability; ...
KQI: N request/sec; M users
App1
App2
App3
SAP-C
SAP-D
Adaptation Function
Service layer

Figure 6.7: Us-Sl reference point

Service Access Points (SAP) are logical end-points describing a customer's location. Connectivity is described as logical links connecting the network functions / applications to each other and to SAP. At this level a SAP is not directly tied to the network (for instance as a specific IP address or switch port), instead they represent attachment points at the service level and may be, for example, a particular branch office identifier, a user name, a group of users, or a connection to another network such as the Internet.

The KQIs attached to network functions and connectivity represent quality goals matching the level (layer) of the service request, such as the number of users handled by a network function, the number of request per second handled, or the total service availability percentage. These KQIs are either calculated from related Key Performance Indicators (KPIs) – compute, storage and networking level performance indicators such as latency, bandwidth, delay, etc.,– or they are KPIs themselves.

However, from information model point of view, the Us-Sl reference point must pass a graph with vertices and edges plus constraints according to arbitrary grouping of connected elements.

6.4.3.3 Sl-Or reference point

The Network Function Forwarding Graph passed through the Sl-Or reference point is a translation of the Service Graph to match the Orchestration Layer representation, at a level of detail suitable for resource orchestration (shown in Figure 6.8). This includes all the components of the Service Graph; network functions / applications are translated/expanded into network functions with known decompositions to corresponding instantiable types exist in the resource orchestration component (for example turning the previously mentioned parental control function into three NFs with internal connectivity); SAPs are translated/expanded into End Points, identifiers meaningful at the network level such as a certain port on a switch, an IMSI or a collection of IP addresses; KQIs are mapped to measurable KPIs and requirements on the ENFs. The KQI mapping may result in insertion of additional network functions into the NF-FG for measuring certain KPIs that cannot be provided in other ways.

The differences in the information passed in the Sl-Or compared to the Us-Sl reference point are that (compound) network functions are decomposed into network functions whose model exists in the orchestrator layer's network function information base and all requirements must be formulated against compute, storage and networking resources, so at Sl-Or only ENFs appear, CNFs get resolved to ENFs.

Otherwise, the graph model (vertices and edges) are the same as at the Us-Sl reference point (see Figure 6.8).

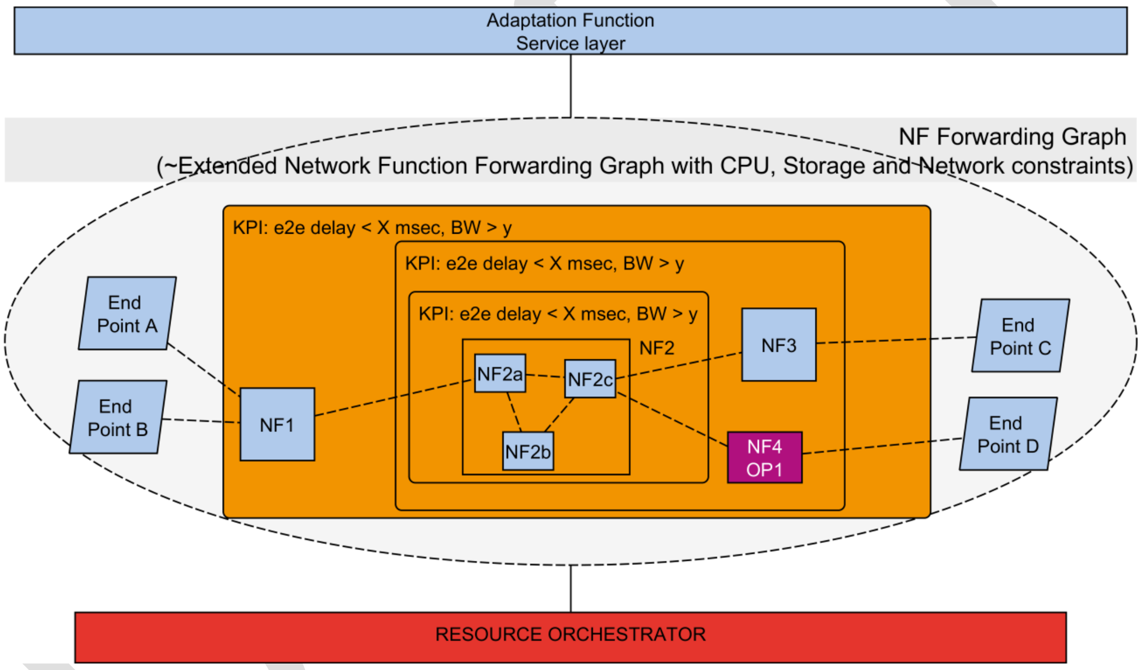

Adaptation Function
Service layer
NF Forwarding Graph
(~Extended Network Function Forwarding Graph with CPU, Storage and Network constraints)
KPI: e2e delay < X msec, BW > y
KPI: e2e delay < X msec, BW > y
KPI: e2e delay < X msec, BW > y
NF2
End Point A
End Point B
NF1
NF2a
NF2c
NF2b
NF3
NF4 OP1
End Point C
End Point D
RESOURCE ORCHESTRATOR

Figure 6.8: Sl-Or reference point

6.4.3.4 Or-Ca reference point

The output of the Resource Orchestrator is an instantiable network function forwarding graph, which, at the model level, corresponds again to a kind of Network Function Forwarding Graph abstraction (see Figure 6.9).

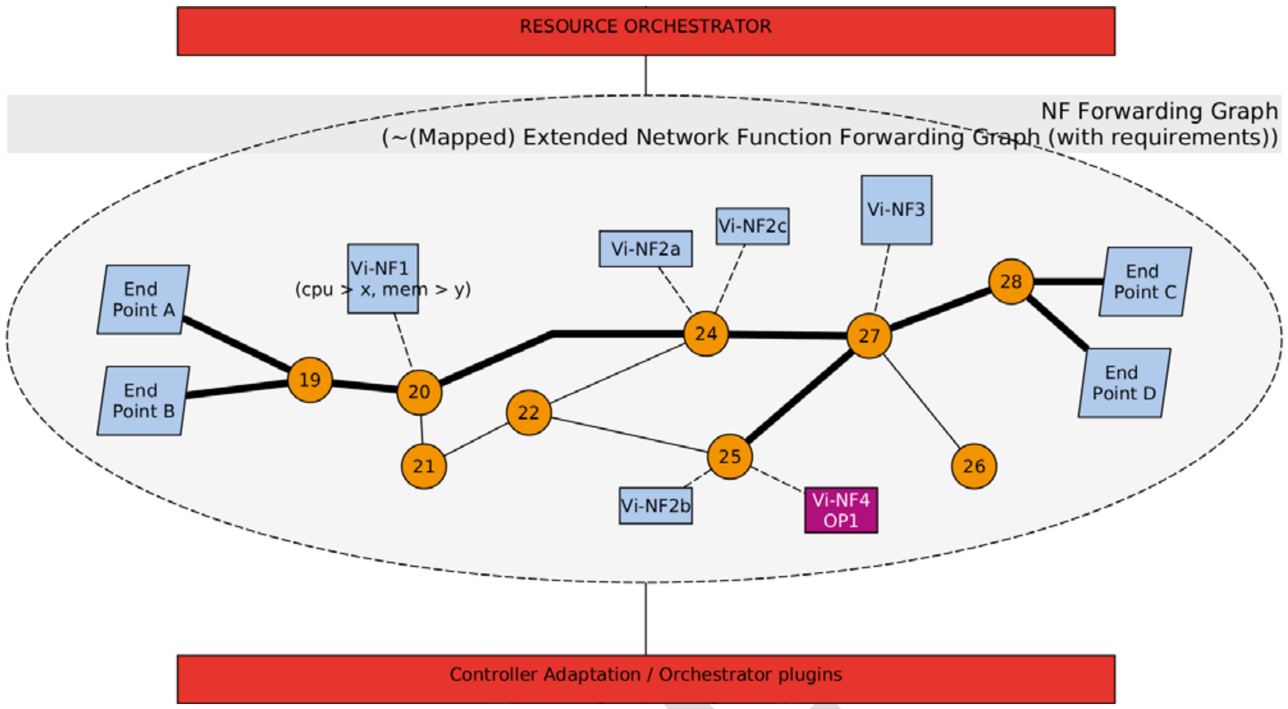


Figure 6.9: Or-Ca reference point

6.4.3.5 Ca-Co reference point

The Ca-Co reference point captures various northbound interfaces related to different controller frameworks for virtualization environments. In UNIFY the reuse and integration of some well accepted managers and controllers for virtualized infrastructure like OpenStack for data centres and OpenDaylight for software defined networks is targeted.

On the other hand, when a Universal Node or an underlying orchestrator is connected to the Controller Adaptation, the same NF-FG representation can be used to define the local scoped service request (NFs and forwarding overlay) and constraints as the output of the orchestration logic. Hence, the Controller Adaptation should only extract and pass the corresponding sub-graph to the UN or the underlying orchestrator.

6.4.4 Monitoring and verification processes

6.4.4.1 Verification process

Enabling run-time verification of code is an important goal of continuous integration as part of the DevOps concept. While traditional DevOps mainly refers to verification of code, this goal in SP-DevOps relates to verification of service definitions and configurations. Automated verification functions on each layer of the architecture facilitate verification as part of each step in the deployment process, allowing identification of problems already early in the service lifecycle. In that sense, verification is less of a process, but rather a set of features providing gatekeeper functions to verify the abstract service models (Service Graphs (SG), Network Function Forwarding Graphs (NF-FG), and resource configuration) before actual instantiation on the Infrastructure Layer takes place.

In the following, the verification functionality will be described on each architecture layer as well as its input and result as depicted in the sequence diagram in Figure 6.10.

- Verification functionality in the Service Layer (SL): a service request from customers/users is received in the form of a Service Graph with a related SLA definition. This abstract Service Graph definition will allow the SL to verify for example the absence of loops and other topological consistency properties like authorization for access to end-point. As a result, Service Graphs can be marked as invalid and returned to the customer/user early on in the deployment process. If the verification does not find any inconsistencies, the deployment process continues as described in section 6.4.3.2. The result of the Service Layer is a NF-FG (derived from the Service Graph), which can as well be verified against absence of loops and other topological properties before being handed over to the Orchestration Layer.
- Verification functionality in the Resource Orchestration component: The resource orchestration component receives a NF-FG and performs the placement of these NFs, resulting in an instantiable network function forwarding graph which can be verified against consistency with respect to the resource, capability and topology descriptions. Furthermore, on this layer the instantiable network function forwarding graph can be verified against policy violations related to placement of NFs and performance impact on already deployed NF-FGs on the chosen infrastructure.
- Verification functionality in the Controller: The Controller components commanding the Infrastructure Layer can consist of a set of different controllers for both compute (for example OpenStack) and networking resources (for instance OpenDayLight). The verification functionality in this Orchestration Layer part will target consistency of specific configuration instances, such as inconsistent network configuration, e.g. in the form of SDN OpenFlow rules.
- Verification and activation in the Infrastructure Layer: The Infrastructure Layer receives configurations via various interfaces in order to instantiate the actual NFs together with their connectivity as well as the necessary Observability Points. Once fully instantiated, the subcomponents of a service need to be verified. As an example, the actual flow tables can be analysed for validity and consistency. Additionally, service activation can be performed through applying fault management tools (see section 6.4.4.2 on observability) both in per-segment and end-to-end fashion.

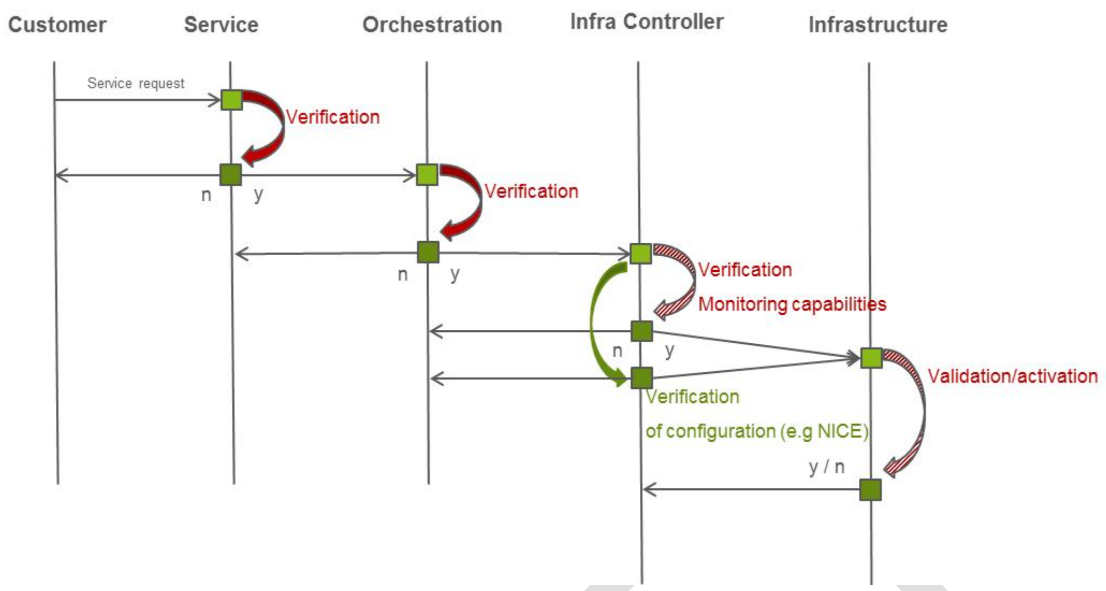


Figure 6.10: Verification Process Flow Chart

6.4.4.2 Observability process

The Observability process is the UNIFY interpretation of the DevOps principle "monitor operational quality" mentioned in section 2.10. It provides visibility onto the operational performance of Service Graphs deployed in the unified production environment. The customer asking for a particular Service Graph to be deployed in the network agrees with the operator of the unified production environment on an SLA, describing a set of Key Performance and Quality Indicators (KPI / KQI) that need to be fulfilled by the environment during the lifetime of the service. The KPI/KQI values expressed in this SLA are the trigger of the SP-DevOps Observability process. The process has two distinct parts:

- A set of translation and tool selection stages
- A set of data generation and processing stages, in which methods for measuring performance parameters of the production environment itself and inspecting the status of traffic forwarded through the Service Graph are used

The translation and tool selection stages determine what should be measured or inspected for a Service Graph, how such measurement or inspection action should be carried out and where such measurement and inspection capabilities are to be instantiated and configured. In addition to parameters related to the Service Graph itself, the Resource Orchestration component may specify KPIs related to the monitoring of the overall production environment that are needed for its internal processes. At the Controller and Infrastructure Layer, these KPIs are processed through the same translation, deployment and instantiation stages as are the service-graph related KPIs. The "Placement", "Deployment" and "Instantiation" stages of the Observability process are required to be dynamic in the sense that they are expected to be triggered following the increase or decrease of resources allocated to a Service Graph or when service KQI requirements are changed on the fly. They are also triggered when Service Graph components are migrated through the production environment. The "Continuous Operation" stage means that the measurement or inspection capability was instantiated and remains in pro-active operation, generating data until its execution is explicitly stopped through another "Instantiation" call. Details of the translation process will be available in D3.1 and D4.2.

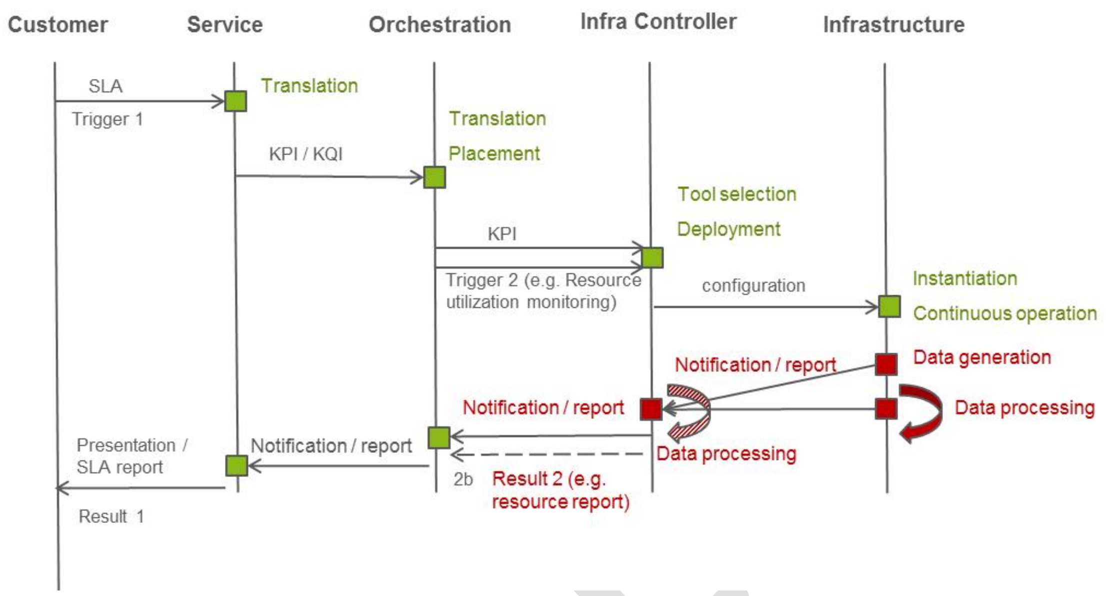


Figure 6.11: Observability Process Flow Chart

Data generation and processing stages of the Observability process rely on novel capabilities for performance and fault management developed by the UNIFY partners. Basic data generation capabilities on any type of resource include counters, notifications and logs generated by the resource itself or the function instantiated and executing on the resource at a particular moment in time. In addition, we present a few examples of data generation capabilities expected to be developed in UNIFY are presented, these includes:

- automated generation of packets testing for reachability conditions for multi-path routing schemes, backup paths and flows that have been inactive for a certain interval
- OpenFlow table analysis tools to find rules that are unreachable or unused
- loss or delay measurement tools that exploit OpenFlow signalling to reduce the load on the production environment

Data processing stages of the Observability process rely on capabilities for filtering and any kind of aggregation (addition, multiplication, subtraction, summarization, etc.) operations that are performed on data generated by managed resources and functions in a unified production environment. Examples of such capabilities expected to be developed in UNIFY include:

- a method for statistical monitoring of flow counters that determines best sampling rates while preserving the accuracy of the information gathered
- an in-network scheme for event correlation
- in-network processing capabilities to aggregate and filter intermediate results for reducing the monitoring's traffic footprint (for example VirtuCast [24])

The ultimate result of data processing stages in the Observability process is delivering a summarized report on KPIs to the initiator of the Service Graph SLA. Results from the data processing stages are consumed also along the components of the architecture, in particular at the Orchestration Layer which relies on the Observability process to supply up-to-date data on the status of the resources of the unified production environment.

6.4.4.3 Troubleshooting process

With reference to troubleshooting, the localization of the source of a problem related to a certain process is meant – in other words, it is about isolating the cause of unexpected behaviour. In UNIFY, automation of troubleshooting will aid processes related to service and VNF development, service fulfilment (for example deployment) as well as service assurance (for instance service operations). In this respect, automated troubleshooting is in-line with DevOps principles such as developing and testing against production like systems as well monitoring and validation of operational quality. Troubleshooting mechanisms will need to operate on several levels of the architecture and will largely take advantage of verification and observability mechanisms and tools introduced as part of the processes described above. These tools will be used collectively for localizing the cause of a problem such as fault or performance degradation. By ruling out sources of the problem the actual root cause can be narrowed down, which might be a rule inconsistency or a physical problem in the infrastructure.

A troubleshooting process is performed either as a request or as an in-network capability implemented in certain functional blocks. Requests can either be made manually by the customer (user), or by automated Service and Orchestration Layer component triggered by subscribed report or notifications provided by the deployed observability points. A requested troubleshooting process aims to follow up on reported bugs, faults, and anomalous states that require further investigation when the faulty or anomalous condition cannot be immediately localized from existing observations and reports. This includes automated deployment and re-deployment of relevant verification and observability functions in order to isolate the root-cause of detected bugs, faults and anomalies.

In-network troubleshooting refers to autonomous fault localization and root-cause capabilities implemented in the observability components in the Infrastructure Layer, meaning that these components exchange relevant information (estimates, log or audit information, states, counter values, etc.) in order to localize certain types of faults and performance degradations to various root-cause conditions in physical or logical devices. Faults and performance degradations that are typically in the scope for in-network troubleshooting are of the kind that the information needed to identify a root-cause is already accessible in the detecting observability component, or can be retrieved by in-network requests between components. Detected and localized faults and performance degradations in the Infrastructure Layer are reported to the controller component and forwarded as necessary to upper layers in the architecture.

Troubleshooting requests and in-network troubleshooting offer complementary functionality that enhance observability and monitoring capabilities needed to achieve the objectives of resource-efficiency and timeliness through a high-degree of automation.

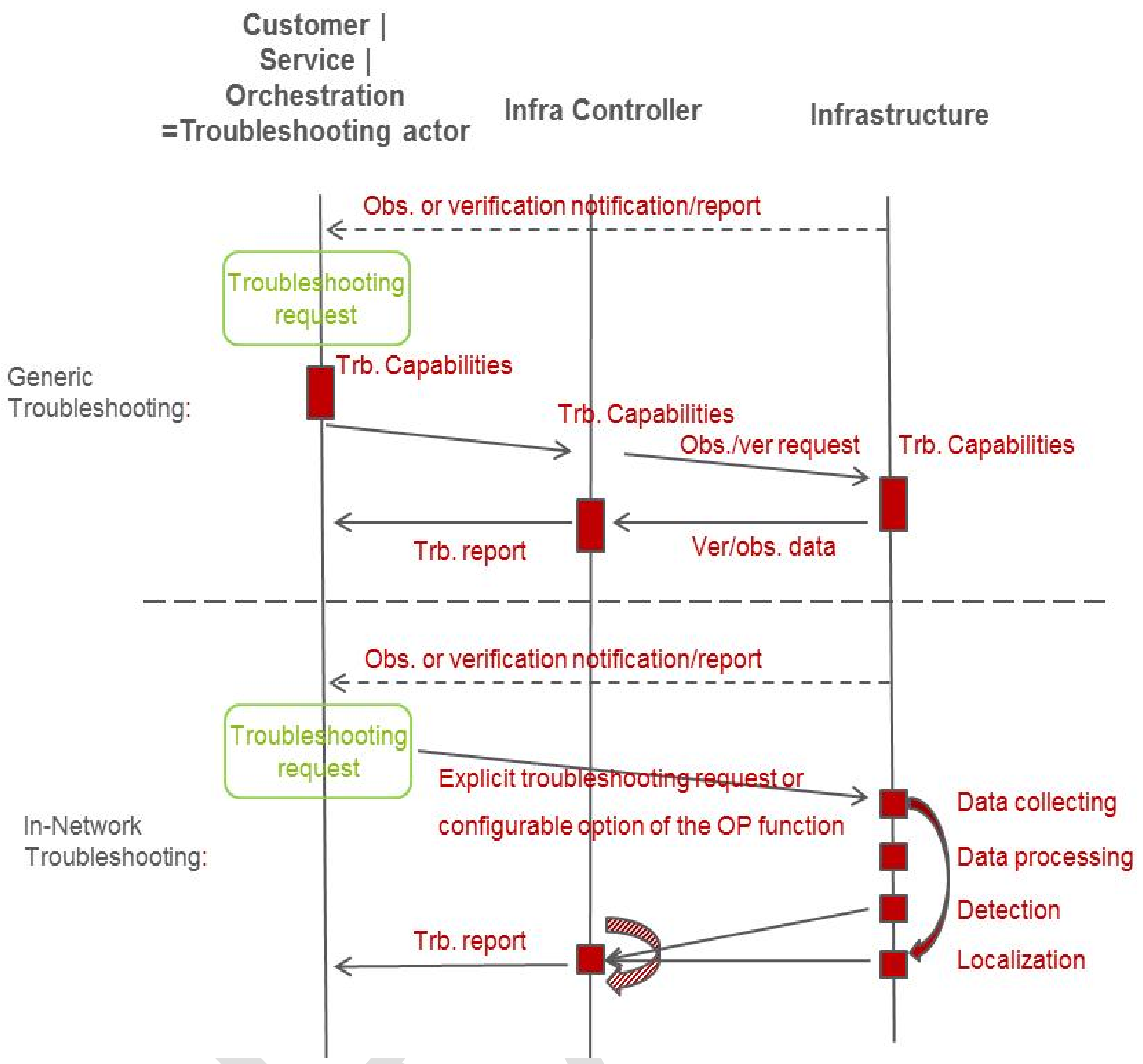


Figure 6.12: Troubleshooting Support Flow Chart

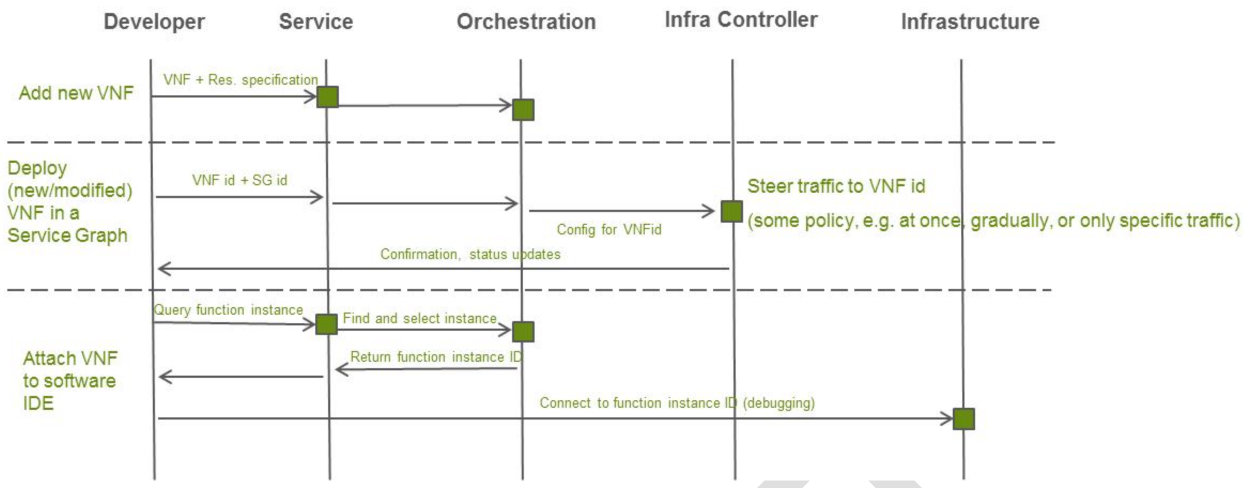
6.4.5 VNF Development support process

When applied in the unified production environment, the VNF Development process supports the team developing functionality for a network function to conform to the DevOps principle "Develop and test against production-like systems" mentioned in section 2.9. The unified production environment provides the means to instantiate a newly developed or updated VNF onto the architecture. It also provides the means to identify the location where a particular instance is being executed, which is important for debugging purposes. The VNF Development process includes only the interactions with entities belonging to the unified production environment. The actual copying of the code to the production environment as well as configuration tasks that need to be performed before and after the instantiation are part of the interaction with the management system and thus not depicted in our process.

Three aspects associated to the VNF development support process are presented:

- Adding a new VNF to the production environment: This sub-process allows developers to add their new VNF (or a new version of an existing VNF) to the production environment for testing and debugging purposes. To start with, the developer submits a description of the VNF capabilities and resource requirements to the Service Layer, which in turn informs the Orchestration Layer about the existence of the new VNF. The Service Layer also stores the description of the new VNF in the service catalogue or inventory. The "adding a new VNF" view is also employed when updating an existing VNF. In this case, the updated VNF is stored in the service catalogue and the Orchestration Layer is informed about any changes in terms of resource requirements that might have been introduced by the update.

- Deploying a new or updated VNF in an existing Service Graph: Besides debugging isolated VNFs, developers can proactively minimize bugs by testing their new VNF code embedded in a complete Service Graph, interacting with other VNFs used in the production environment. Here, the developer announces their intention to deploy a particular VNF (identified through a VNF id) in a particular Service Graph (identified through a Service Graph id) instantiated in the unified production environment. The request is received at the Service Layer, forwarded to the Orchestration Layer which determines what the resources are allocated to the current instance of the VNF and what resources need to be allocated to the new instance. The Controller responsible for the new resources is announced and it configures the policies associated to steering traffic towards the new VNF instance. Once the policies are in place, the developer is informed about the availability and location of the resources and can proceed with the actual copying of the code, which takes place outside the VNF Development process.
- Attaching VNF to software Integrated Development Environments (IDE): This sub-process is an enabler for actual VNF debugging activities. The developer queries the Service Layer by providing a Service Graph id and the type of the VNF in order to determine where the instance they are interested in is being executed. The Service Layer forwards the request to the Resource Orchestration component, which determines what resources are used for the VNF instance and provides the developer with an identifier for the resources and the VNF instance. The developer can then connect to the running VNF instance by means of tools such as distributed software debuggers, developed outside of UNIFY, and perform the debugging activity. The assumption of the "Attach VNF to software IDE" view is that no special needs have to be fulfilled in order to enable to connect to the VNF using a software debugger. When special functionality needs to be deactivated or modified (for example, when the VNF instance is protected by a firewall), the developer needs to submit to the Service Layer a request to change the Service Graph accordingly.



Developer
Service
Orchestration
Infra Controller
Infrastructure
Add new VNF
VNF + Res. specification
Deploy (new/modified) VNF in a Service Graph
VNF id + SG id
Config for VNFid
Steer traffic to VNF id (some policy, e.g. at once, gradually, or only specific traffic)
Confirmation, status updates
Query function instance
Find and select instance
Return function instance ID
Attach VNF to software IDE
Connect to function instance ID (debugging)

Figure 6.13: Processes supporting Development of VNFs

6.5 Management

In a telecommunications operator's environment, network devices and functions are capable of remote configuration and management. There are special groups and standardisation bodies that study the management of network systems. The Tele Management Forum (TMF) in particular is studying ways of achieving telecom management systems based on Tele Management Network for service providers, see section 2.9 for details.

In UNIFY the network functions and devices have an interface (often referred to as a "northbound" interface) to a management function. This management function is often highly complex and composed of a great many distributed component parts. Responsibility of the management layer is to configure and monitor the network functions.

In detail, there are four different types of management interfaces present:

- General purpose and customer centric management systems and interfaces, like policy systems, BSS, customer configuration interfaces, etc.
- Management functions researched and developed by UNIFY for programming and monitoring the network (see section 6.4.3, 6.4.4 and 6.4.5 for details)
- Management functions and interfaces for virtual and physical network functions, which are under control of another entity and interface with the UNIFY framework for instance optimisation requests, provisioning tasks or management information.
- Management of everything which is not covered by the UNIFY framework, architecture and developments like detailed hardware information on temperature, physical infrastructure monitoring. These systems might have interfaces to the UNIFY architecture as well.

In UNIFY, a "resource" is a unit of production that has compute, network or storage capabilities. In contrast, TMForum eTOM defines as "resource" as any capability that is being used for supporting a particular service. In this respect, network functions or VNFs from the UNIFY vocabulary are seen as "resources" from an eTOM framework perspective. Being aware of the difference in the definitions is key to understanding the split between the management and orchestration functionality in the UNIFY framework.

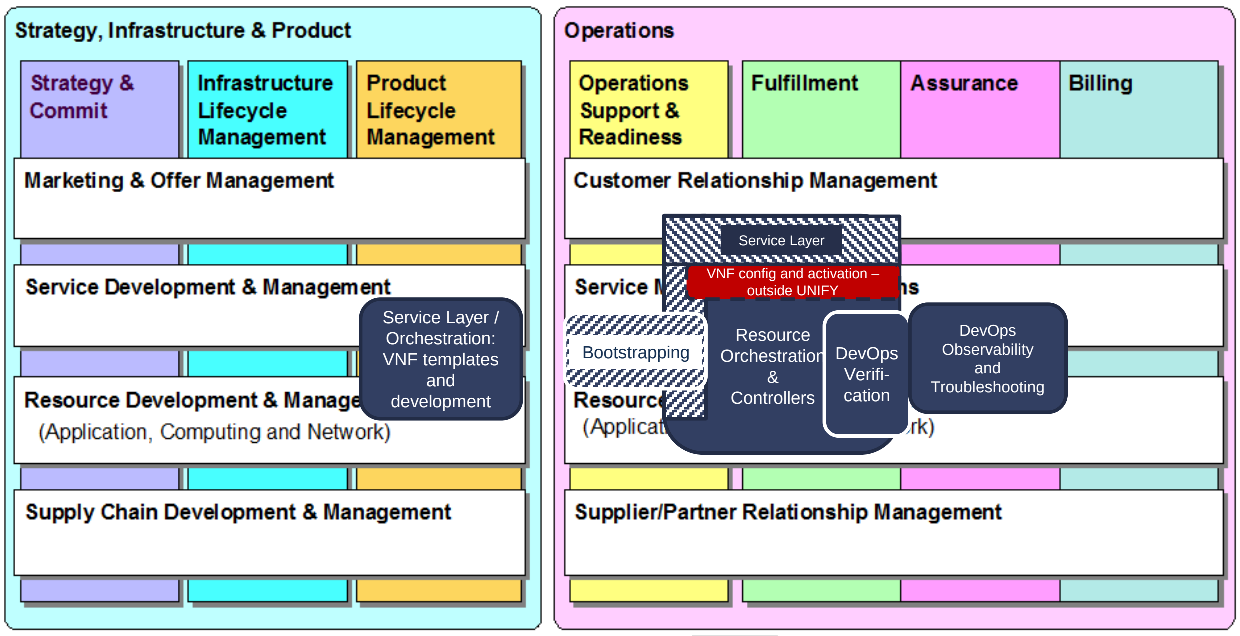
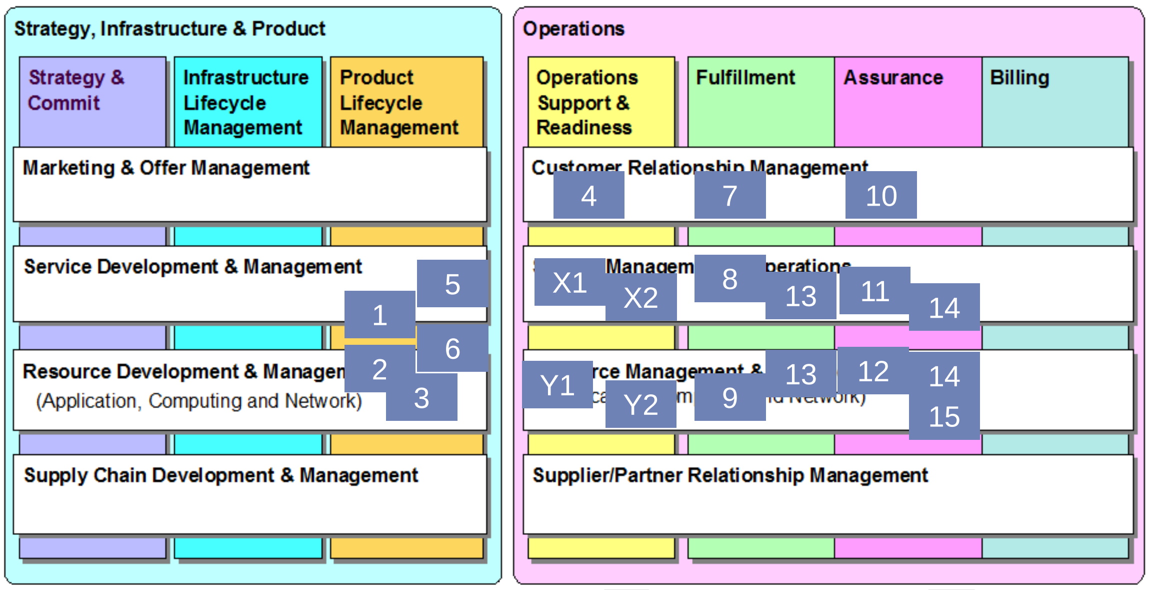

Strategy, Infrastructure & Product
Strategy & Commit
Infrastructure Lifecycle Management
Product Lifecycle Management
Marketing & Offer Management
Service Development & Management
Resource Development & Management (Application, Computing and Network)
Supply Chain Development & Management
Service Layer / Orchestration: VNF templates and development
Operations
Operations Support & Readiness
Fulfillment
Assurance
Billing
Customer Relationship Management
Service Layer
VNF config and activation – outside UNIFY
Bootstrapping
Resource Orchestration & Controllers
DevOps Verifi-cation
DevOps Observability and Troubleshooting
Supplier/Partner Relationship Management

Figure 6.14: Relationship of UNIFY and TMForum

The TMForum eTOM framework defines, by means of the Product and Infrastructure Lifecycle Management – Resource Development and Management, a series of processes for planning and providing enough resources for the network functions (see section 2.9 and 5.6 for details). Typically this involves significant workloads associated to determining the locations where certain network functions should be placed as physical boxes, determining the right dimensioning parameters for these network functions in accordance to traffic and load models, acquiring and taking delivery of the hardware. In the UNIFY production environment, the Orchestration Layer components greatly simplify and automate these eTOM processes. The customer or an equivalent system is left with determining the right dimensioning constraints for the (virtual) network functions and then submitting a request to the Orchestration Layer.

The eTOM framework also defines a series of processes associated to Resource Management and Operations in the Operations Support and Readiness, Fulfillment and Assurance dimensions. In UNIFY, the part of these processes associated with Readiness and Fulfillment for UNIFY resources (compute, storage, network) is covered by the Orchestration and Controller components of the architecture, except verification-related processes. The UNIFY DevOps Verification processes perform part of the verification tasks associated to the Readiness and Fulfillment eTOM components. Assurance processes for the production environment are supported in part by the UNIFY DevOps Observability and Troubleshooting.


Strategy, Infrastructure & Product
Strategy & Commit
Infrastructure Lifecycle Management
Product Lifecycle Management
Marketing & Offer Management
Service Development & Management
Resource Development & Managem
(Application, Computing and Network)
Supply Chain Development & Management
1
2
3
5
6
Operations
Operations Support & Readiness
Fulfillment
Assurance
Billing
Customer Relationship Management
4
7
10
X1
X2
8
13
11
14
Y1
Y2
9
13
12
14
15
Supplier/Partner Relationship Management

1	Service Development (NF-FG templates)	11	NF-FG Service Problems & Quality == KPI
2	VNF / Atomic VNF Development	12	NF & Network Resource Problems & Quality
3	VNF developer support	13	Verification
4	Defining App or service store for Users	14	Monitoring and observability
5	Defining service manifests (KPI vs resources)	15	Troubleshooting
6	Defining VNF resource manifests	X1	Network Function and manifest data base
7	Service Request (Service Graph)	X2	Configuration and providing interfaces to monitoring and trouble-shooting information
8	NF-FG KPI provisioning	Y1	Collecting capabilities and resource information, topology discovery
9	NF & Network Resource Provisioning	Y2	Configuration and providing interfaces to configuration, observability elements
10	SLA management		

Figure 6.15: Mapping management aspects of TMForum to UNIFY

The part of Readiness and Fulfillment processes connected to the VNFs is outside the UNIFY scope and expected to be delivered through the TMForum-compliant management systems associated to the particular network function. This means that the customer for which a VNF graph is instantiated needs to deploy their own management systems in order to configure the VNF, as well as monitor and troubleshoot faults and performance management issues at the VNF level. These management systems will receive monitoring and event information from the UNIFY production environment by means of the DevOps Observability processes.

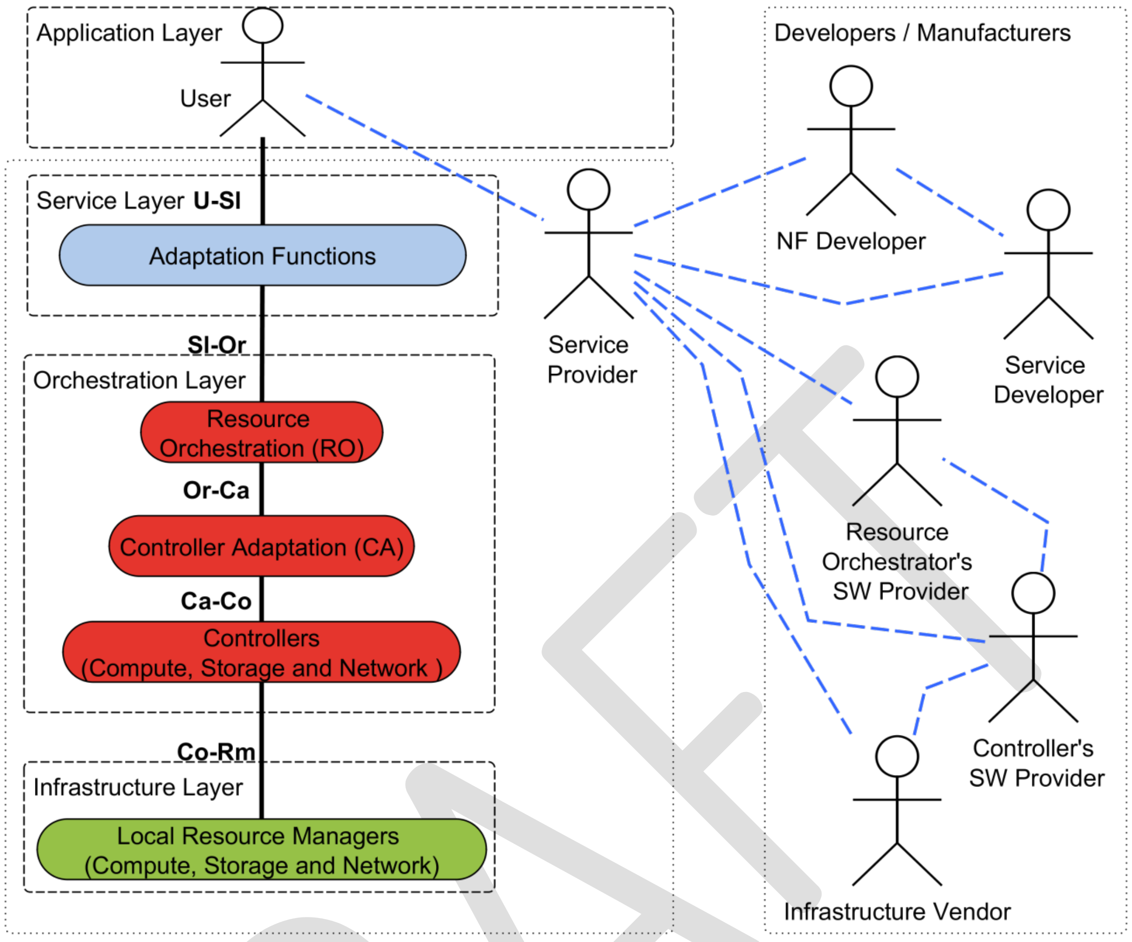
They will also need to integrate information generated by the DevOps Troubleshooting process to support the Assurance - Resource Trouble Management eTOM process.

The Figure 6.15 maps these management aspects of the TMForum eTOM to UNIFY.

6.6 Initial definition of potential business roles

Looking at the different layers of the proposed architecture, one can identify actors who potentially can contribute with different components to create a virtualization and orchestration framework up to the users. In the following, key actors and their relations are identified which will play a role in a value chain.

- User: Users consume communication and cloud services. Users can be residential or enterprise end users, but also can be other service providers (multi domain setup), over the top (OTT) providers or, content providers. Users sign contracts with the service provider for specific services with service level agreements (SLA).
- Service Provider: Service providers offer services to users subject to specific SLAs. Service providers make direct use of logical resource management (from Orchestration SW Providers) and data plane and virtualization management (from Controller SW Providers). Service providers access the resources via a resource manager functionality of an infrastructure provider. Service providers could be in the role of a user with respect to other service providers.
- Orchestration SW Provider: Software developers (like vendors, 3rd party) who create software systems and functions (services, libraries and apps) to manage the global view of abstract resources.
- Controller SW provider: Software developers (for instance open source communities, vendors or 3rd parties) developing data plane managers (e.g., OpenFlow) and cloud managers (like OpenStack) to present abstraction of the underlying resources (networking and cloud).
- Infrastructure Vendor: Providers of physical resources including networking, compute and virtualization environments.
- NF Developers: internal to the service provider or third party developers who designs, develops and/or maintains network functions. The SP DevOps framework shall support the development directly for the NF developer or indirectly through the service provider itself.



Application Layer
User
Service Layer U-Sl
Adaptation Functions
Sl-Or
Orchestration Layer
Resource Orchestration (RO)
Or-Ca
Controller Adaptation (CA)
Ca-Co
Controllers (Compute, Storage and Network)
Co-Rm
Infrastructure Layer
Local Resource Managers (Compute, Storage and Network)
Service Provider
Developers / Manufacturers
NF Developer
Service Developer
Resource Orchestrator's SW Provider
Controller's SW Provider
Infrastructure Vendor

Figure 6.16: Potential business roles

6.7 Mapping of UNIFY Architecture to Industry Maps

In principle UNIFY also works on the orchestration functions of the ETSI NFV framework. In this case ETSI's forwarding graphs would appear as inputs to the UNIFY Orchestration Layer logic, which maps network functions to logical and physical resources of the infrastructure (see Figure 6.17).

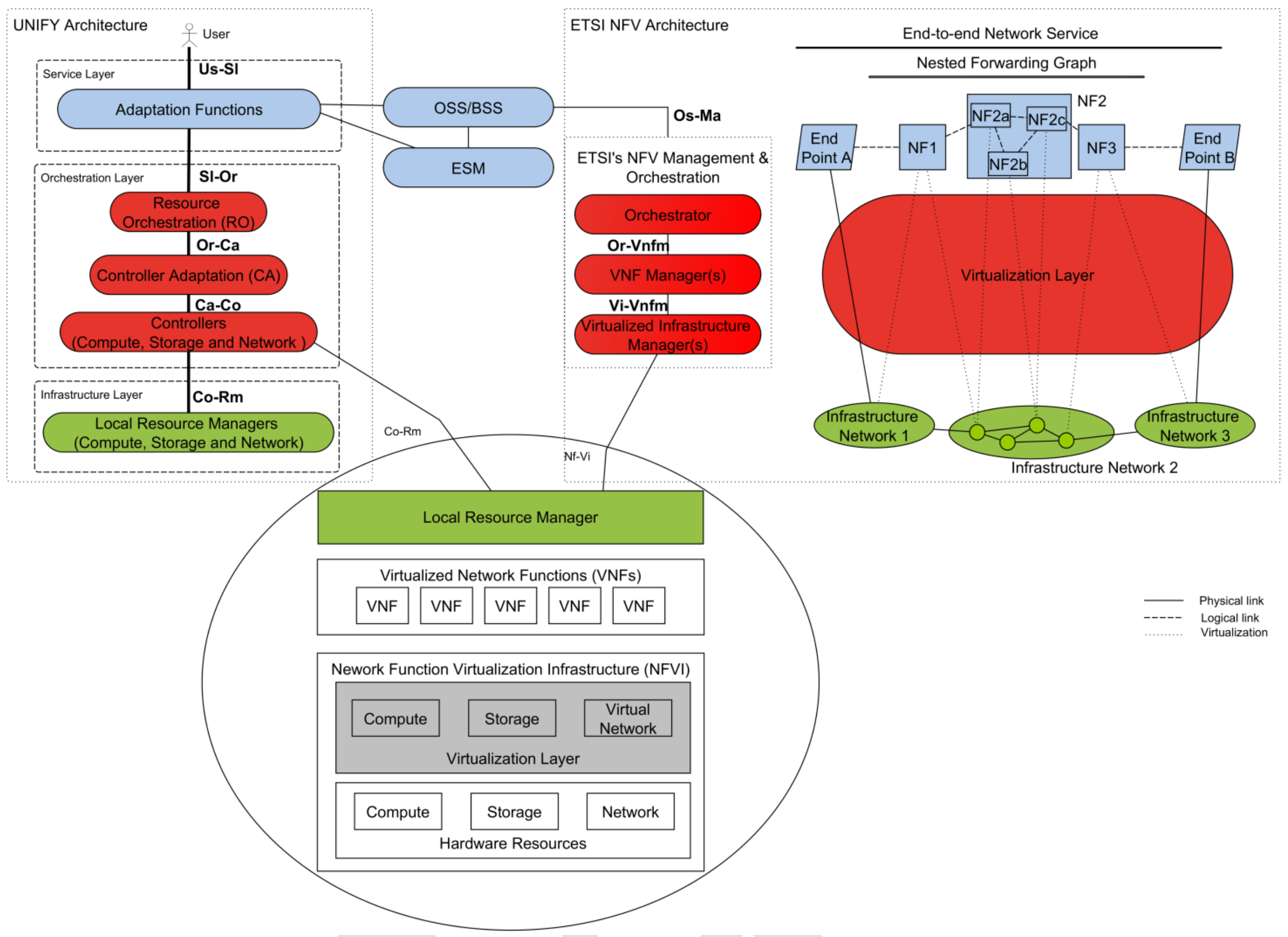


Figure 6.17: UNIFY vs. ETSI NFV architecture

ONF recently published a solution brief [3] positioning SDN and NFV. Compared to ONF's landscape shown in Figure 6.18, UNIFY considers an Orchestration Layer expanding to the basic enabler for network function virtualization articulated in orchestration functions beyond controllers' functionality. In addition UNIFY defines business applications as network function chains in the Service Layer compared to ONF's application layer.

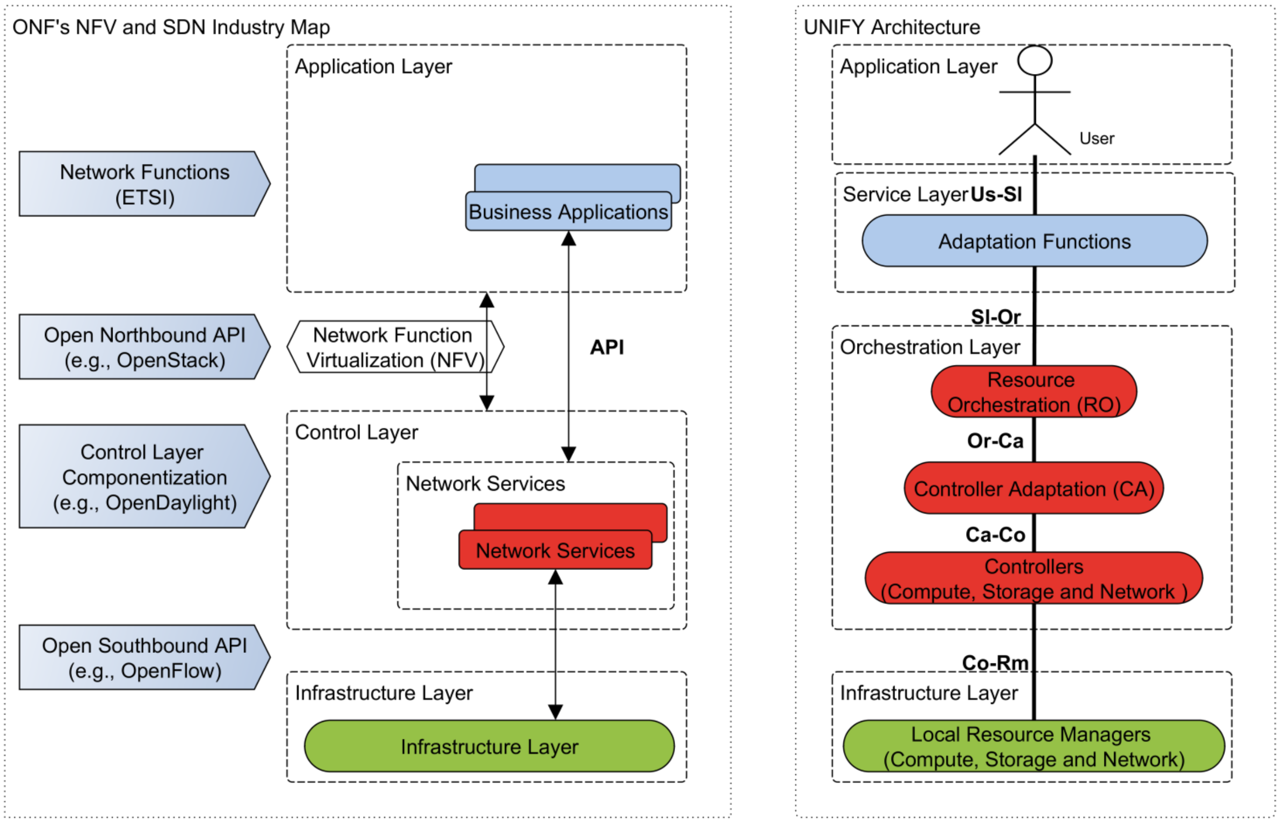

ONF's NFV and SDN Industry Map
Application Layer
Network Functions (ETSI)
Business Applications
Open Northbound API (e.g., OpenStack)
Network Function Virtualization (NFV)
API
Control Layer
Control Layer Componentization (e.g., OpenDaylight)
Network Services
Network Services
Open Southbound API (e.g., OpenFlow)
Infrastructure Layer
Infrastructure Layer
UNIFY Architecture
Application Layer
User
Service Layer
Us-Sl
Adaptation Functions
Sl-Or
Orchestration Layer
Resource Orchestration (RO)
Or-Ca
Controller Adaptation (CA)
Ca-Co
Controllers (Compute, Storage and Network)
Co-Rm
Infrastructure Layer
Local Resource Managers (Compute, Storage and Network)

Figure 6.18: UNIFY vs. ONF's NFV & SDN industry map

7 Functional Architecture

7.1 Introduction

The overarching architecture was the first design step for the UNIFY architecture (see section 6). The different high-level layers and interfaces have been defined in section 6.2. These architecture layers have been illustrated in section 6.3 adding understanding and two real-world use cases. The different processes have been applied to the architecture as well (section 6.4) providing an idea how the architecture works for certain, individual processes. A special focus was put on management (section 6.5). Nonetheless, the different architectural and functional elements have not been combined and presented in a comprehensive way.

All aspects are now jointly presented in this section; illustrating the interactions between the different elements, the information exchange between the layers and therefore combining the different UNIFY concepts for the unified production environment. As a result, an initial definition of the interfaces (section 7.2) and a first comprehensive set of systems and sub-systems of the different layers (section 7.3) are presented.

7.2 Initial definition of interfaces

The top-level functional models including the layers of the overarching UNIFY architecture and the functions relevant to the high-level interfaces between the layers are shown in Figure 7.1.

The main interfaces are repeated here for completeness: the application (representing an user) - service layer (Us-Sl), the service layer – resource orchestration in the Orchestration Layer (Sl-Or), the controller adaptation-controller interface as Orchestration Layer internal interface (Ca-Co), the controller in the Orchestration Layer to the Infrastructure layer interface (Co-Rm) and the Resource Orchestration – application specific resource control block interface (Cf-Or) (see Figure 6.1 and Figure 7.1 for reference). All interfaces crossing a layer are detailed with their functional aspects in the following paragraphs.

The interface between the application and the service provider is at the highest abstraction level in the UNIFY architecture. The corresponding reference point is denoted by Us – Sl. Via this interface, a end user or another service provider (e.g., retail provider, OTT provider, content provider, etc.) is able to request a given service from the service provider. This service is described by a high-level Service Graph with pre-defined service-level parameters. This Service Graph can be, for example, the output of a top level GUI where the customer is able to configure services in a user-friendly way. Reports on high-level KPIs have to be transmitted from the service provider to the customers.

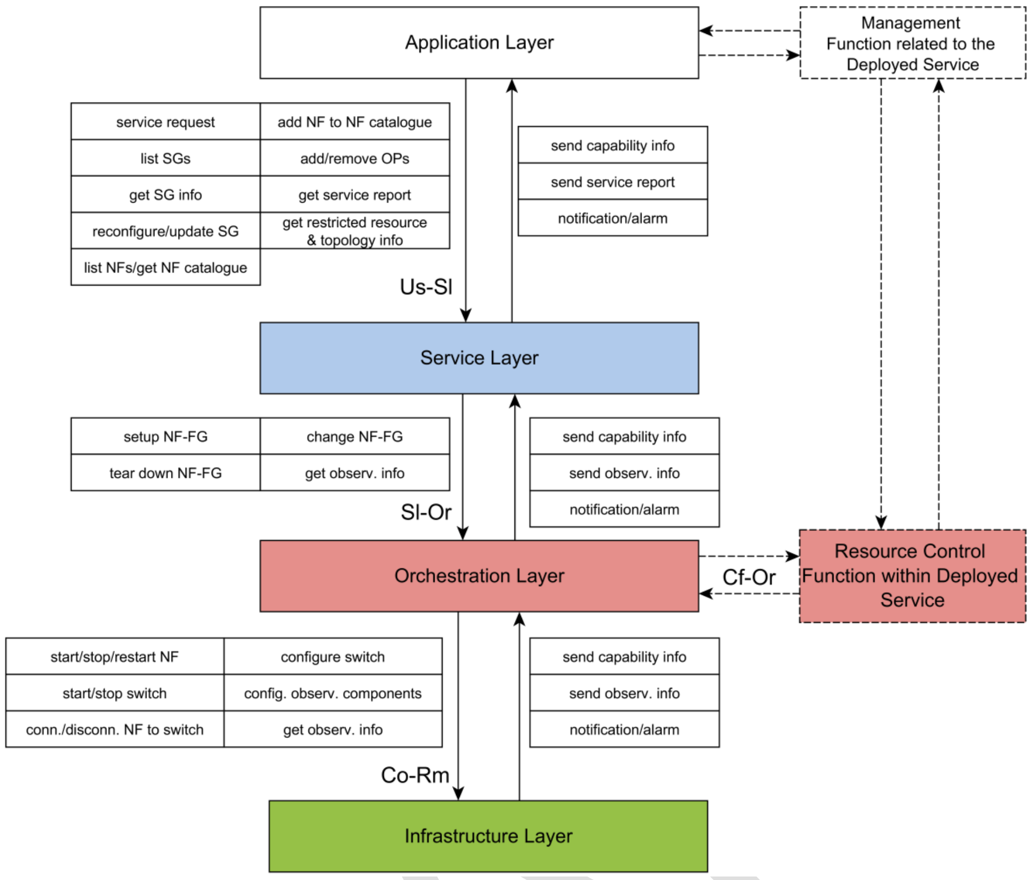


Figure 7.1: Top level functional model

The Us – Sl interface / API has to support the following operations:

- service request
- list Service Graphs
- get Service Graph info
- reconfigure/update Service Graph
- list NFs / get NF catalogue
- add new / modified NF to NF catalogue
- add/remove observation point
- get /send service report
- get restricted resource & topology information
- send capability information of orchestration layer to application layers
- notification / alarm

The Service layer forwards a transformed Service Graph, namely Network Function Forwarding Graph (NF-FG), to the Orchestration Layer via the Sl – Or interface. This data structure contains information about the required functions, capabilities as well as needed information by resource optimization tasks performed in orchestrator modules. For programmability details on the data structure and carried information, see Section 6.4.3.3.

The Sl – Or interface / API has to support the following operations:

- setup of NF-FG
- tear down of NF-FG
- change of NF-FG
- get / send observability information
- notification / alarm
- send capability information about the network functions, abstracted resource information representing the available resource capabilities

The orchestration layer is the most complex part of the UNIFY architecture consisting of several functional sub-layers, modules and interfaces between them (e.g. Or – Ca or Ca – Co) which need to be detailed and documented in D2.2. The connection towards the infrastructure is realized via southbound interfaces. Here, several available protocols can be invoked and adapted to our special purpose architecture. For example, available protocols, such as OpenFlow, NETCONF, Libvirt, can be used within the Co – Rm interface.

Co – Rm interface / API has to support the following operations:

- start/stop/restart VNF
- start/stop switch (forwarding element)
- connect/disconnect VNF to switch
- configure switches
- configure observability components
- get / send observability information
- notification / alarm
- send capability information

It should be noted that the architecture has potentially an application specific resource control function as shown in Figure 6.1. This element is under investigation and will be further detailed in D2.2.

7.3 Initial definition of systems / sub systems

The main part of the design is identifying the relevant and necessary functional blocks and assigning clear, separate responsibilities to them. Besides this identification, the appropriate naming of the components at all abstraction levels is also an important and sometimes challenging part of the process. This section is devoted to introducing the identified functional blocks at different layers and the relations between them.

The architecture requires information stored in databases. The different abstractions and translations demand mapping information about the different capabilities. This information needs to be available at each architectural layer in the required granularity. The boot-strapping process for example attaches describing information with elements and initialises different databases, e.g. topology or configuration part of the resource database. It is left open for implementation, whether these databases are implemented separately at each layer or as one for the whole system. Given that the architecture is designed for recursion, partitioning and layering, separated ones or instantiated with a limited view seem to be the logical consequence.

Another common issue across the architectural layers are monitoring, observability and verification. This requires a chain of elements, which adapt to the appropriate abstraction at the each layer and an information exchange between the layers.

Access to the different modules is currently not defined. The programming and monitoring are supposed to be top-down or bottom-up processes, which require a trusted relationship between the different functional elements. How exactly access and modifications in the UNIFY architecture are taken care is left for further investigations.

7.3.1 Service Layer

The Service Layer, being at the top level of the UNIFY architecture, is the main interacting point between the customer and the service provider. The Us-Sl reference point is described in the previous section and further details are already discussed in section 6.4.3.2.

The main functional components of the service layer are the following (see Figure 7.2):

- Service Abstraction
- Service Graph Abstraction

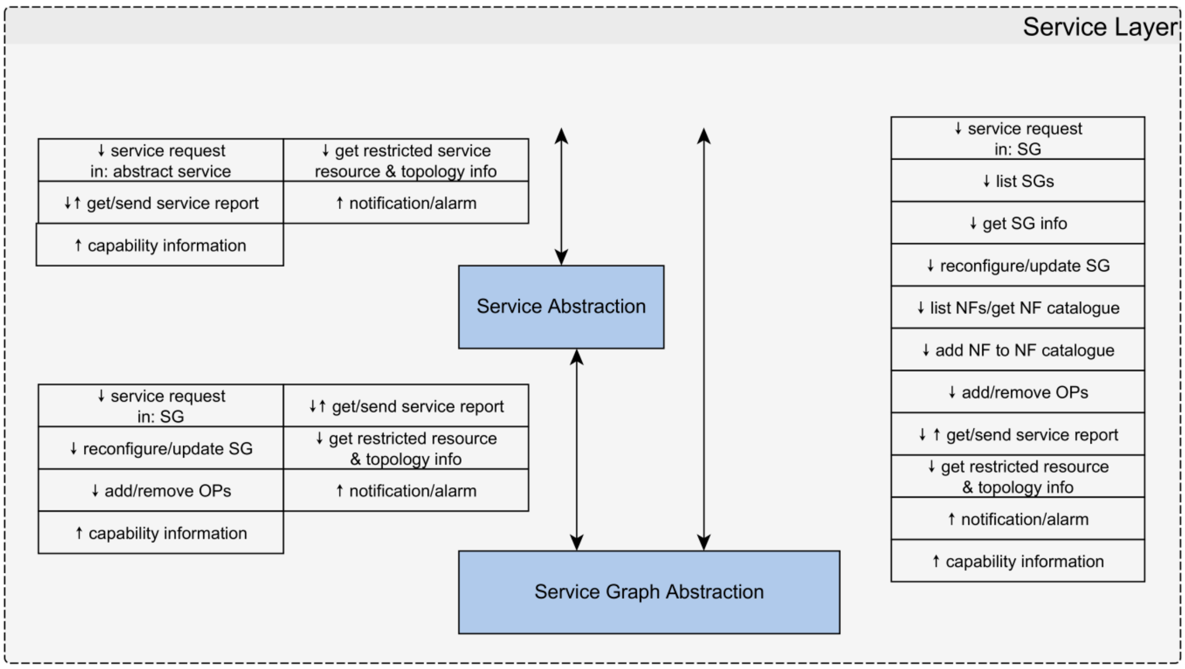


Figure 7.2: High level structure of the service layer

The Service Abstraction module is responsible for processing high level service requests from customers (receiving e.g., from a Web GUI), translating the service description into Service Graph, deriving KQIs from high level SLAs or service classes, sending resource, topology and other capability information and generating service reports and notifications in case of relevant events. This module can be further decomposed into sub-blocks as it is shown in Figure 7.3.

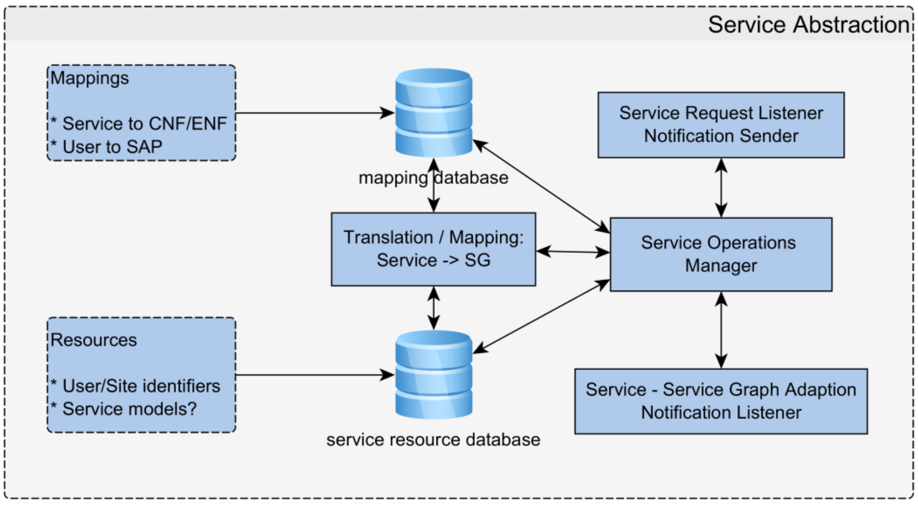

Service Abstraction
Mappings
* Service to CNF/ENF
* User to SAP
mapping database
Service Request Listener Notification Sender
Translation / Mapping: Service -> SG
Service Operations Manager
Resources
* User/Site identifiers
* Service models?
service resource database
Service - Service Graph Adaption Notification Listener

Figure 7.3: Service Abstraction module

Service Abstraction module has two communication interfaces with corresponding functional blocks: Service Request Listener/Notification Sender communicates with the higher layers (Web GUI or other type of clients), while Service – Service Graph Adaption / Notification Listener is responsible for handling messages from/to the Service Graph Abstraction module. This type of decomposition concerning the communication blocks is used in subsequent components as well. The main operation of Service Abstraction module and its sub-component is controlled by the Service Operations Manager. It can instruct the Translation / Mapping function to translate the received abstract service requests into a Service Graph. For this operation, it makes use of the mapping and the service resource database, the first one of them storing information and models needed for translation between abstract service description and SGs, and the second one containing information on service resources, such as user/site identifiers, restricted topology view and resource information.

The next component of the service layer is the Service Graph Abstraction module which is responsible for handling service requests described by Service Graphs, managing Service Graphs (listing current Service Graphs, returning information on requested and reconfiguring Service Graphs, etc.). It is also capable of managing NF catalogue / NF repository, and Observation Points (OPs), and generating reports and notifications concerning Service Graph performance as well as transmitting capability information about components of the Service Graph. On the one hand, the Service Graph Abstraction module communicates with the Service Abstraction module which translates between abstract service description as outcome of the top-level GUI and formal SGs. On the other hand, Service Graph Abstraction module is able to cooperate with "lower-level" entities handling Network Function Forwarding Graphs and feeding them directly into the system. The structure and decomposition of the Service Graph Abstraction module is presented in Figure 7.4.

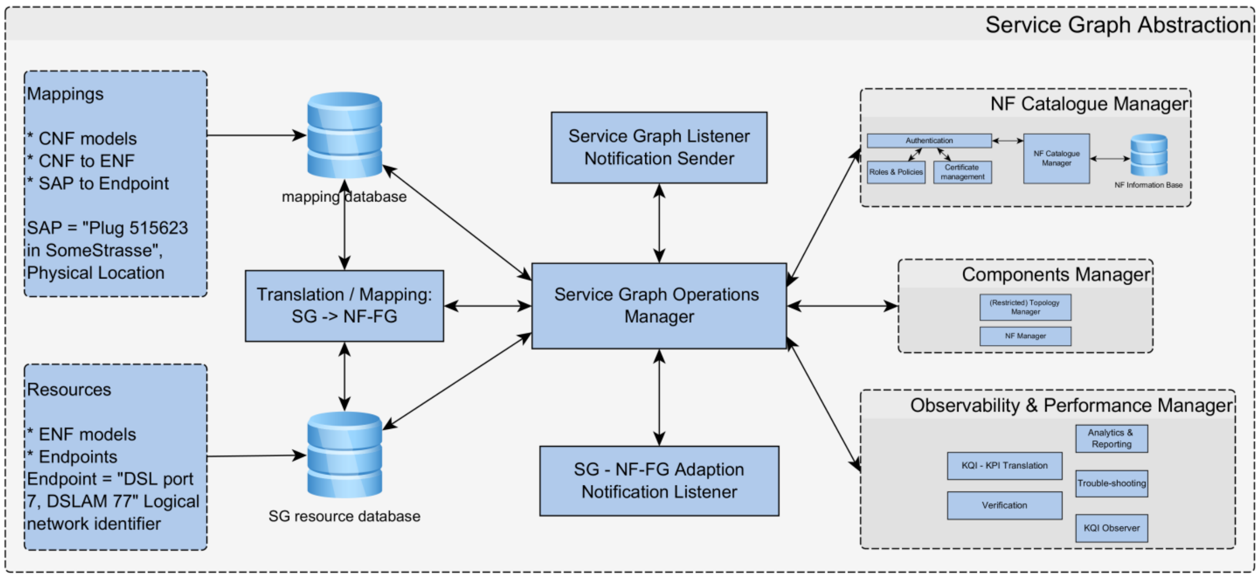

Service Graph Abstraction
Mappings
* CNF models
* CNF to ENF
* SAP to Endpoint
SAP = "Plug 515623 in SomeStrasse", Physical Location
mapping database
Service Graph Listener Notification Sender
NF Catalogue Manager
Authentication
Roles & Policies
Certificate management
NF Catalogue Manager
NF Information Base
Translation / Mapping: SG -> NF-FG
Service Graph Operations Manager
Components Manager
(Restricted) Topology Manager
NF Manager
Resources
* ENF models
* Endpoints
Endpoint = "DSL port 7, DSLAM 77" Logical network identifier
SG resource database
SG - NF-FG Adaption Notification Listener
Observability & Performance Manager
Analytics & Reporting
KQI - KPI Translation
Trouble-shooting
Verification
KQI Observer

Figure 7.4: Service Graph Abstraction module

The logical structuring of Service Graph Abstraction module is very similar to the structure of the previously seen Service Abstraction part. There are two blocks responsible for communication with upper and lower parts of the system, respectively. Here, these components are called Service Graph Listener / Notification Sender and SG – NF-FG Adaption / Notification Listener. The main director is the Service Graph Operations Manager which coordinates all other functional components internally and holds all information about a dedicated service graph. At this sub-layer, Service Graphs have to be mapped to NF-FGs by a dedicated Translation / Mapping function which can use and interact with a mapping database and a SG resource database. The first one contains mapping information necessary for SG – NF-FG translation including compound network function

(CNF) models, mapping of compound NFs to elementary ones (CNF → ENF), conversion between service attachment points and physical endpoints (SAP → endpoint). In addition, required application specific control blocks are discovered and added to the network function forwarding graphs. The second database stores Service Graph resource information. This database is based on restricted views from the Orchestration Layer on a per-client basis.

Service Graph Abstraction module contains three additional blocks. NF Catalogue Manager is responsible for all tasks related to the Network Function Catalogue (or NF database) including access management, database updates (add a new NF), NF requests, etc. This information could have different sources like initial updates during boot-strapping of network functions. In order to keep the restricted topology information up-to-date and to provide NF management for e.g. the Resource Control Function of a deployed service, the Components Manager as a distinct module is introduced. Finally, the Observability & Performance Manager is a dedicated component handling tasks related to observation, performance monitoring, verification, and troubleshooting of SGs. It is also responsible for KQI reporting. The reporting and required translation between KQI and KPI is supported by and transmitted via the Service Graph Operations Manager.

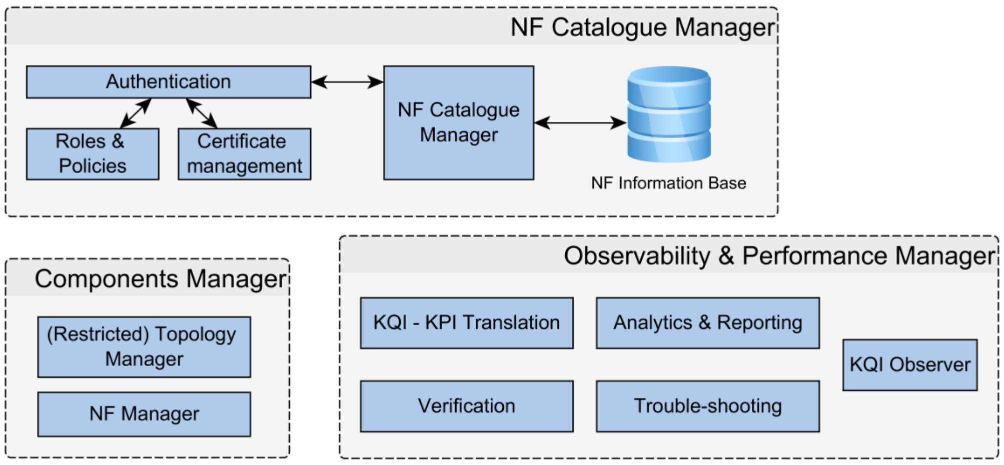


Figure 7.5: Additional components of Service Graph Abstraction module

Further decomposition of these additional components (NF Catalogue Manager, Component Manager, Observability & Performance Manager) can be seen in Figure 7.5.

7.3.2 Orchestration Layer

The Orchestration Layer has three sub-layers with well separated responsibilities. These are:

- Resource Orchestration layer
- Controller Adaptation layer
- Controller Layer

In the following functional description, the Controller Adaptation will be combined with the Resource Orchestration Layer into the combined Resource Orchestration and Controller Adaptation Layer (ROCAL) for abstracting from internal processes already discussed in section 6.

ROCAL is responsible for maintaining a global view of the infrastructure resource topology of a domain as well as mapping the NF-FGs coming from the Service layer to resources of that topology to NF-FGs coming from the Service layer, taking into account the associated KPIs they have and more general optimization goals. If required the ROCAL deploys application specific control blocks (which are also a particular type of network functions) for determining required resources or allowing steering of resource requests directly from the application. The outcome of the process is an per client or request adopted, restricted Network Function Forwarding Graph. As part of the mapping of NF-FG to the resources, the ROCAL determines where to place:

- Elementary Network functions (for instance choosing an already existing VNF/PNF or the node where a new VNF should be created)
- Application specific control blocks like Resource Control Functions
- Logical network configuration (mapping the logical links/connectivity of the NF-FG)
- Placement of Observability Points necessary to capture the KPIs (both provided by lower layers and instantiated as NFs)

The high-level functional view of the ROCAL is shown in Figure 7.6 while details on relevant sub-components (Decomposition/Mapping manager, Global resource manager and Observability & Performance manager) are given in Figure 7.7.

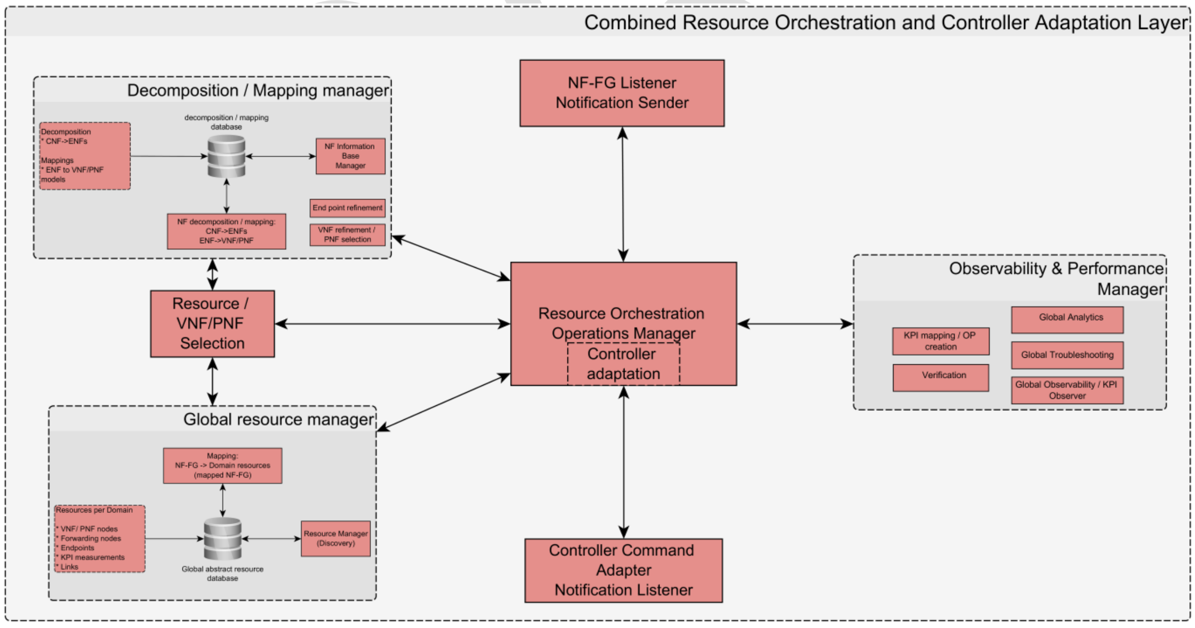

Combined Resource Orchestration and Controller Adaptation Layer
Decomposition / Mapping manager
decomposition / mapping database
Decomposition * CNF->ENFs Mappings * ENF to VNF/PNF models
NF Information Base Manager
End point refinement
VNF refinement / PNF selection
NF decomposition / mapping: CNF->ENFs ENF->VNF/PNF
NF-FG Listener Notification Sender
Resource / VNF/PNF Selection
Resource Orchestration Operations Manager
Controller adaptation
Observability & Performance Manager
KPI mapping / OP creation
Verification
Global Analytics
Global Troubleshooting
Global Observability / KPI Observer
Global resource manager
Mapping: NF-FG -> Domain resources (mapped NF-FG)
Resources per Domain * VNF/ PNF nodes * Forwarding nodes * Endpoints * KPI measurements * Links
Global abstract resource database
Resource Manager (Discovery)
Controller Command Adapter Notification Listener

Figure 7.6: High-level functional model of the combined Resource Orchestration and Controller Adaptation Layer

The ROCAL has similar communication elements as one has seen in case of the Service layer. NF-FG Listener / Notification Sender manages the connection to the Service layer, while Controller Command Adapter / Notification Listener communicate with the Controller Layer. The central entity is the Resource Orchestration

Operations Manager responsible for operating the NF-FG and coordinating with all involved elements. It has a dedicated component for adapting different types of network- and DC controllers. Decomposition / Mapping manager encompasses functions which are essential to decompose NFs based on NF Information Base and to map elementary NFs to virtual or physical ones (ENF → VNF/PNF mapping models). It also has sub-components, such as NF decomposition / mapping, VNF refinement / PNF selection, End point refinement module, and a NF Information Base Manager. The next main component is the Global resource manager which is responsible for keeping global resource database up-to-date, mapping NF-FGs to global resources (mapped NF-FG), optimizing resource utilization, and automatically discovering resources. A specific function, namely Resource / VNF/PNF Selection allows control/configure mapping decisions. Finally, the Observability & Performance Manager deals with performance monitoring, observation, KPI measurement on a global view. It provides global analytics to upper layers performs troubleshooting tasks and triggers the creation of new OPs if necessary. The Controller Adaptation block provides complementary functionality to the Resource Orchestrator. It is responsible for splitting NF-FG into sub graphs, identifying the responsible controllers, sending and forwarding request to the controllers in appropriate formats. For monitoring issues, there will be similar components as shown in Figure 7.7.

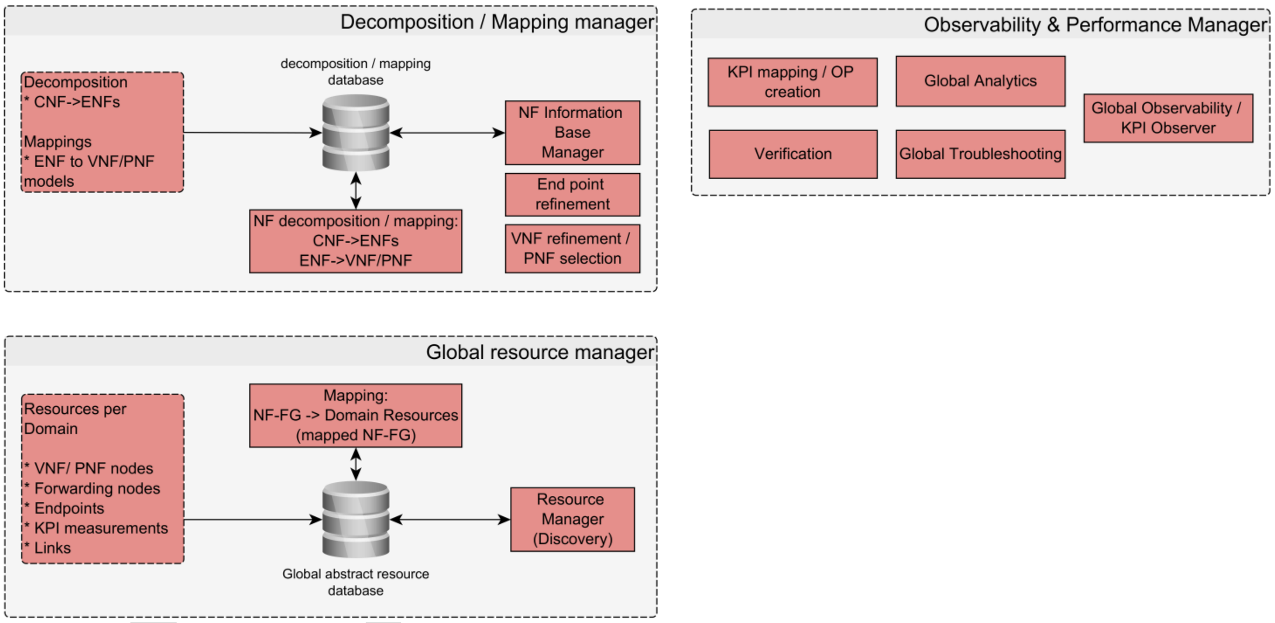
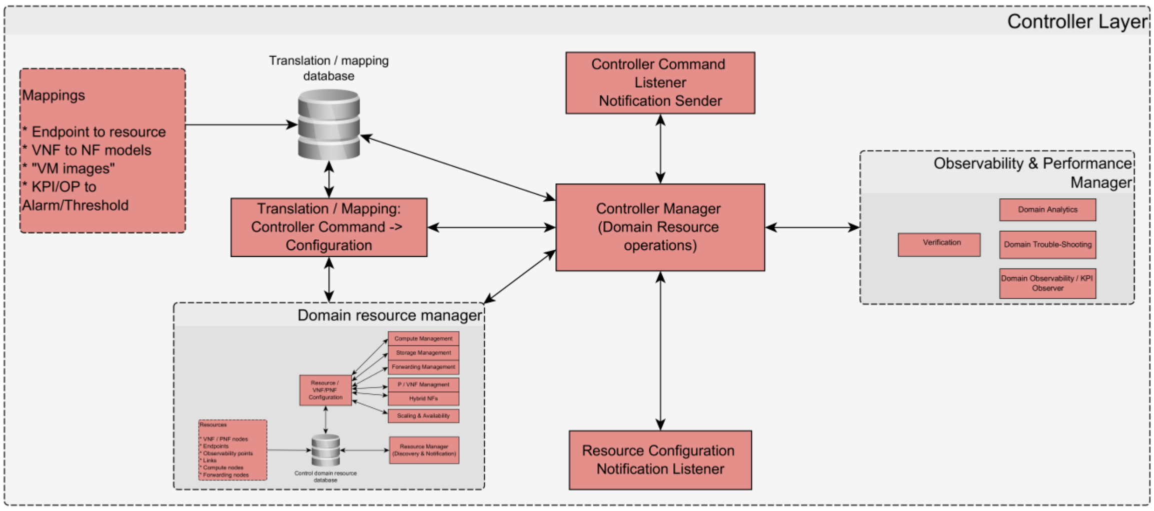

Decomposition / Mapping manager
decomposition / mapping database
Decomposition * CNF->ENFs Mappings * ENF to VNF/PNF models
NF Information Base Manager
End point refinement
VNF refinement / PNF selection
NF decomposition / mapping: CNF->ENFs ENF->VNF/PNF
Observability & Performance Manager
KPI mapping / OP creation
Global Analytics
Global Observability / KPI Observer
Verification
Global Troubleshooting
Global resource manager
Resources per Domain * VNF/ PNF nodes * Forwarding nodes * Endpoints * KPI measurements * Links
Mapping: NF-FG -> Domain Resources (mapped NF-FG)
Resource Manager (Discovery)
Global abstract resource database

Figure 7.7: Details of relevant sub-components of ROCAL

The Controller Layer is responsible for detailed control and management of the different infrastructure resources and virtualization systems of a specific domain. It also provides aggregated or filtered information to the Orchestration layer (topology updates, aggregated measurements, etc.). Based on the adopted, potentially part of NF-FG, this component determines implementation of NFs and network connectivity best among NFs and service attachment points (e.g. adding forwarding entries to existing P2P connections or creation of new connections). The outcome consists of detailed configurations which can be used by the Infrastructure Layer to physically instantiate all necessary components and setup corresponding network connections.



Controller Layer
Mappings
* Endpoint to resource
* VNF to NF models
* "VM images"
* KPI/OP to Alarm/Threshold
Translation / mapping database
Controller Command Listener Notification Sender
Translation / Mapping: Controller Command -> Configuration
Controller Manager (Domain Resource operations)
Observability & Performance Manager
Domain Analytics
Verification
Domain Trouble-Shooting
Domain Observability / KPI Observer
Domain resource manager
Resource Configuration Notification Listener

Figure 7.8: Decomposition of Controller Layer

The high-level structure of the Controller Layer is presented in Figure 7.8. The communication with upper and lower layers is handled similarly as other modules (Controller Command Listener / Notification Sender and Resource Configuration / Notification Listener). The director function is realized by the Controller Manager. This logic controls the operation of a domain by coordinating the sub-modules. Again a Translation / Mapping module is responsible for translating mapped NF-FGs (given as controller command, for instance via a REST API) into detailed domain specific configuration. For storing translation information, a specific database is introduced as well. The tasks of low-level resource control are carried out by the Domain resource manager. The details of this component are shown in the leftmost part of Figure 7.9. Domain resource manager encompasses several sub-manager components, such as Compute Management, Storage Management, Forwarding Management, P/VNF Management, Hybrid NFs, Scaling & Availability. It also contains a database storing information on control domain resources, and a Resource / VNF/PNF Configuration logic.

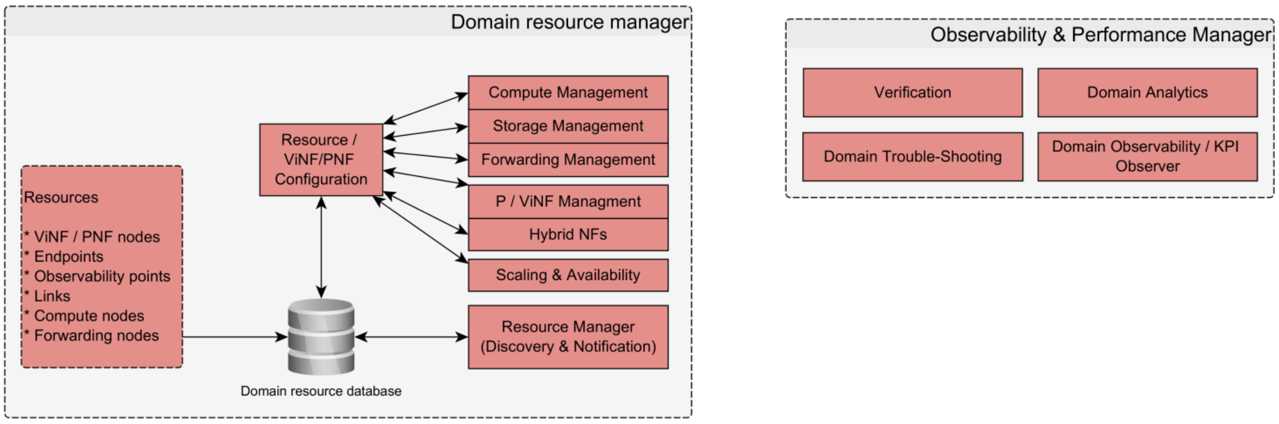


Figure 7.9: The Domain resource manager and the Observability & Performance manager

The Observability & Performance manager is responsible for domain specific observation, monitoring, troubleshooting, verification and reporting. The sub-components are shown in the rightmost part of Figure 7.8.

7.3.3 Infrastructure Layer

The Infrastructure Layer is the lowest layer of the UNIFY architecture including different types of physical resources. These diverse set of resources are handled in a unified way by our Orchestrator system. At his layer, network, compute and storage resources constitute the basis of all provisioned services.

The functional decomposition of the Infrastructure Layer with the most important components is shown in Figure 7.10.

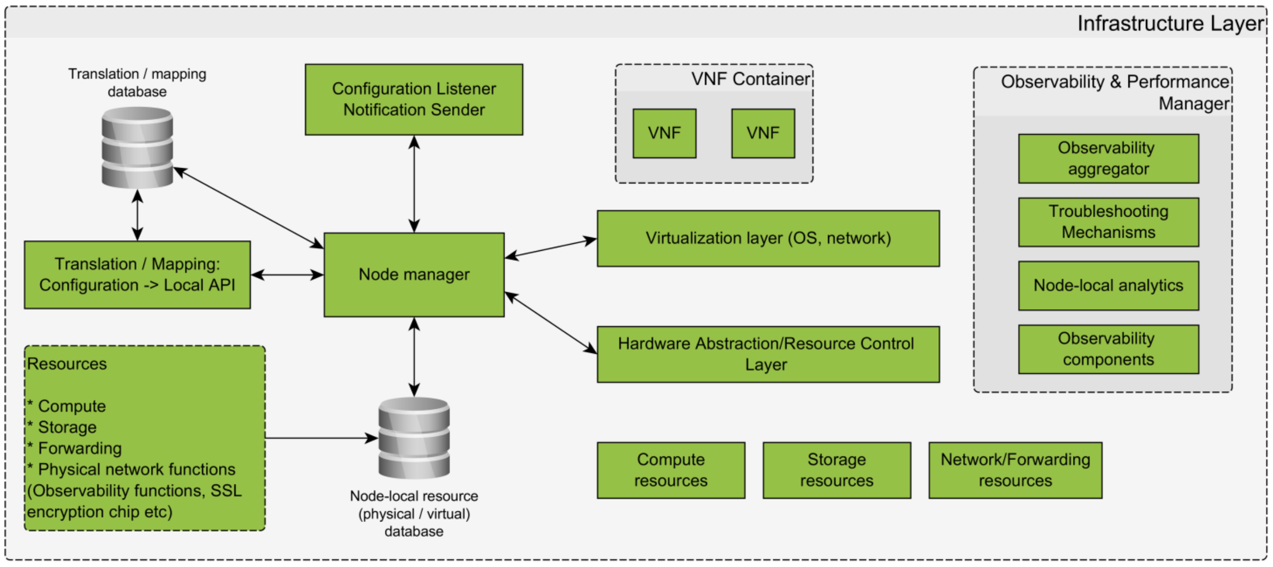

Infrastructure Layer
Translation / mapping database
Configuration Listener Notification Sender
VNF Container
VNF
VNF
Observability & Performance Manager
Observability aggregator
Troubleshooting Mechanisms
Node-local analytics
Observability components
Translation / Mapping: Configuration -> Local API
Node manager
Virtualization layer (OS, network)
Hardware Abstraction/Resource Control Layer
Resources
* Compute
* Storage
* Forwarding
* Physical network functions
(Observability functions, SSL encryption chip etc)
Node-local resource (physical / virtual) database
Compute resources
Storage resources
Network/Forwarding resources

Figure 7.10: Functional model of the Infrastructure Layer

The main component of the Infrastructure layer is the Node manager which is a node-local management agent. It is responsible for the communication with the outside world via the Configuration Listener / Notification Sender (if no local orchestrator is available unlike in the Universal Node – see section 6.3) and also responsible for managing all aspects of the node. In practice this module can consists of different modules like in the UN (see D5.2) and is strictly dependent on the implementation.

There is a Translation / Mapping function with a dedicated database which converts the incoming configuration information according to local APIs. Information on node-local resources is also stored in a database which can be read and written by the Node manager. The physical resources, such as compute, storage and networking/forwarding resources are accessed via a Hardware Abstraction / Resource Control Layer which is controlled by an hypervisor. VNFs are run in special execution environment (VNF Containers) which definitely depend on the selected node's environment (Universal Node, Data Center, etc.). The Observability & Performance Manager deals with node-local part of observation and performance management.

8 Conclusions

This deliverable is the first comprehensive documentation of the UNIFY architecture concluding the task "*to propose and analyse use case scenarios and to define the UNIFY reference architecture*".

As such the deliverable summarizes:

- the results of state of the art analysis
- description of use cases improving today's telecom processes and operations
- definition of design, functional and business requirements
- architecture principles reflecting the different design dimensions
- two of three steps of the UNIFY reference architecture definition, namely description of the overarching and of the functional architecture.

One major starting point was the analysis of the state of the art provided by the research community, the standardisation organisations, industry initiatives and best practices (see section 2). This resulted in a list of 30 important aspects, which are grouped in three major blocks of general recommendations for the development, design principles and architecture aspects. Overall these aspects give an additional first guidance on how and what to develop. Beside typically technology focused initiatives within IETF and ETSI, a focus of this analysis was on the careful study of process frameworks and principles of the TMForum and the DevOps community.

The second starting point was the development, analysis and discussion of use cases (see section 3). In the first round eleven use cases have been selected which represent different aspects of converging carrier network scenarios. This set of use cases has been grouped into three clusters:

- "Infrastructure Virtualization" is showing node internal aspects with the deployment and operation of virtual network functions
- "Flexible Service Chaining" is representing the combination of different physical and virtual network functions inside a domain
- "Network Service Chain Invocation for Providers" is concentrating on multi domain operation inside one entity or across different providers

Aspects of monitoring, verification and support of virtual network functions development are orthogonal and described in each group or cluster. For practical reasons, two selected use cases have been further detailed and will be used throughout the project. The "Elastic Network Function" use case covers the operation of virtual network functions on Universal Nodes, there as the "Secure & Content aware IP VPN" use case concentrate on the operational and management activities in an unified production environment combining private and public resources for optimal service deployment.

Reflecting state of the art and based on the use cases, the first results of the technical developments for the areas of programmability framework, of network management and of the DevOps concept as well as Universal

Node concept, a set of 46 requirements has been designed (see section 4). 28 requirements are topics raised by the technical work packages of UNIFY in a feedback process. The remaining 18 requirements describe overarching business, high-level architecture and framework requirements as well as operational requirements. Overall, the desired behaviour and capabilities are defined with this set of requirements. Moreover the list completes input to the architecture design task, too. The 28 technology specific requirements will be further detailed and prioritized in the individual technical work packages.

Related architectures have been designed before the UNIFY project already started and therefore especially the best practices and existing developments, but although the use case groups and requirements were used to define a base line for the design principles and general properties of the UNIFY architecture (see section 5). Another important philosophic aspect must be the possibility for agility and stability in the design phase. Therefore a three step approach was used, starting with a high-level overarching architecture, refining into more fine granular components, sub-components and interfaces in the functional architecture and finally describing the detailed components in the system architecture. In addition, aspects of narrow-waist desire, layering, virtualisation, abstraction, recursiveness and a simplified processes model are the foundation of the architecture design.

The first step of the architecture design, the overarching architecture, used this set of design principles and combined everything into a three-layered architecture, depicted below (detail see section 6).

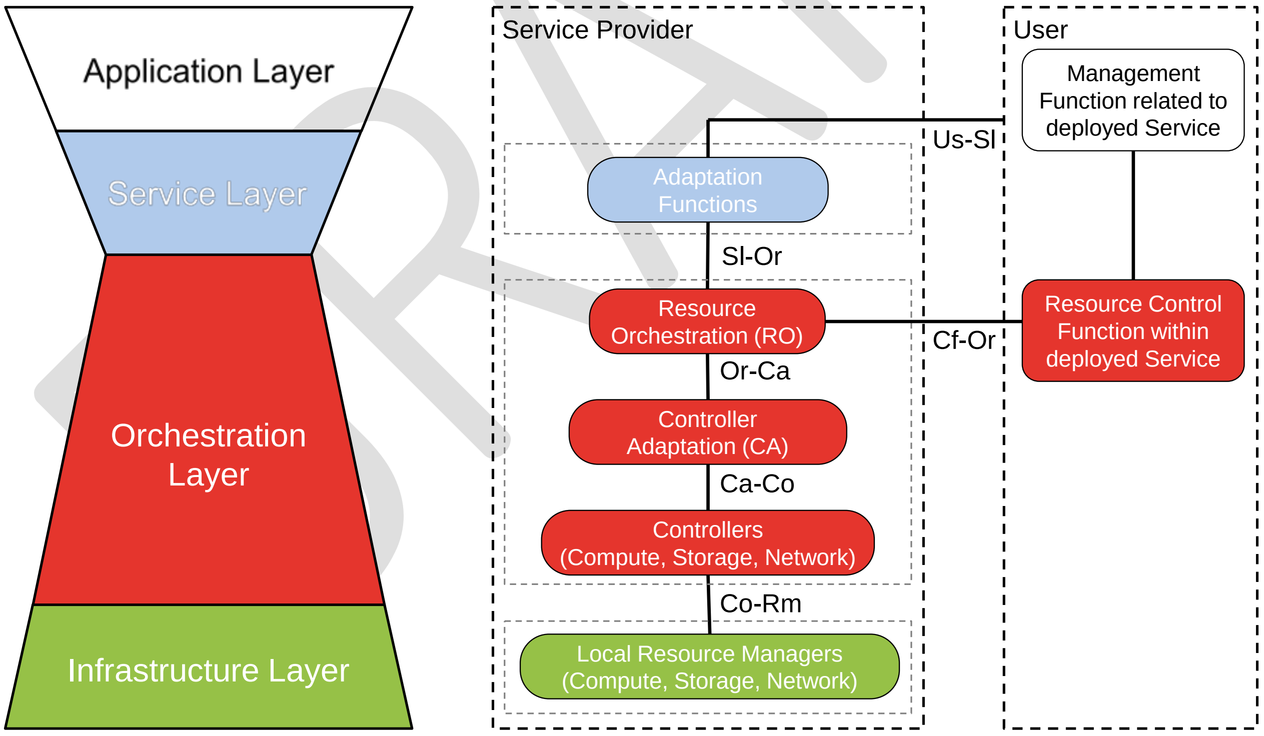

Application Layer
Service Layer
Orchestration Layer
Infrastructure Layer
Service Provider
User
Management Function related to deployed Service
Us-Sl
Adaptation Functions
Sl-Or
Resource Orchestration (RO)
Cf-Or
Resource Control Function within deployed Service
Or-Ca
Controller Adaptation (CA)
Ca-Co
Controllers (Compute, Storage, Network)
Co-Rm
Local Resource Managers (Compute, Storage, Network)

Figure 8.1: The three layered UNIFY architecture

The overarching architecture consists of the three layers, service, orchestration and infrastructure. On top, an application layer represents the user and is not part of the architecture itself. In addition, two "planes" are identified, a service provider plane representing the kernel of the architecture and a user space responsible for running the applications, but also feeding resource information into the kernel. The service layer is responsible for providing translation between user demand and technical descriptions. The orchestration layer has three responsibilities, which could be also perceived as sub-layers:

- Resource Orchestration analyses service requests and tries to find the most appropriate resources. It is supported by the Resource Control Function, which provides continuous feedback on the required resources of applications
- Controller Adaptation provides an adaptation between resource requests and the dedicated controllers of the specific resource
- Controllers are entities responsible for the logical control of specific resources

Local Resource Managers in the Infrastructure Layer are managing entities on the physical infrastructure, like an OpenFlow agent on a switch. In parallel, each of the layers is responsible for providing and operating data related to the specific service request like describing templates or monitoring information. Beside this general approach, detailed description of the six processes Boot-strapping, Programmability, Verification, Observability, Troubleshooting and VNF Development support with respect to the three layers. As a result a set of functional elements were identified. In addition, an analysis of examples including management, technical and business role aspects show the applicability of the UNIFY three-layered overarching architecture in the real-world and verify the gap analysis.

The second design step is the definition of the functional architecture (see section 7) that combines the layered overarching architecture with the process analysis and details resulting functional components including databases and interface properties. An important point here is the similarity in-between the layers, for example each layer has a dedicated component on observability and performance management with the responsibilities for providing reports and analysis as well as trouble-shooting and verification. The dominant data structure between the components is the network function forwarding graph (NF-FG), a construct which represents virtual and physical network functions (on different detailed levels), end points and connections in-between components.

Finally, the foundations of the UNIFY architecture for the unified production environment are set and already discussed and agreed in-between the technical work packages. As next step, the functional architecture will be revised by the technical work packages and the system architecture, that represents the third design step, will be designed. The results will be documented in the deliverable D2.2.

References

[1] S. Shenker, "A Gentle Introduction to Software Defined Networks," Technion, [Online]. Available: http://tce.technion.ac.il/files/2012/06/Scott-shenker.pdf.

[2] „Open Networking Foundation Homepage,“ [Online]. Available: https://www.opennetworking.org/.

[3] O. N. Foundation, 13 05 2014. [Online]. Available: https://www.opennetworking.org/working-groups/working-groups-overview.

[4] „ETSI ISG NFV,“ [Online]. Available: http://www.etsi.org/deliver/etsi_gs/NFV.

[5] CIMI Corporation, 2013. [Online]. Available: http://www.cloudnfv.com/WhitePaper.pdf.

[6] T. M. Bohnert and A. Edmonds, "D2.2 Overall Architecture Definition, Release 1," http://www.mobile-cloud-networking.eu/, 2013.

[7] FP7 ICT SPARC, "Project Website," [Online]. Available: http://www.fp7-sparc.eu/. [Accessed 27 06 2014].

[8] W. John, "SPARC D3.3 Split Architecture for Large-Scale Wide-Area Networks," [Online]. Available: http://arxiv.org/abs/1402.2228. [Accessed 30 6 2014].

[9] „IETF SFC Group,“ [Online]. Available: https://datatracker.ietf.org/wg/sfc/.

[10] „IETF NVO3 Group,“ [Online]. Available: https://datatracker.ietf.org/wg/nvo3/.

[11] „IETF I2RS Group,“ [Online]. Available: http://datatracker.ietf.org/wg/i2rs/.

[12] „IRTF SDN Research Group,“ [Online]. Available: http://irtf.org/sdnrg.

[13] [Online]. Available: http://api.openstack.org.

[14] [Online]. Available: http://www.tmforum.org/CurrentRelease135/15582/home.html. [Accessed 14 05 2014].

[15] S. Sharma, DevOps for Dummies - IBM Limited Edition, Hoboken, New Jersey: John Wiley & Sons, Inc., 2014.

[16] “Broadband Forum (BBF),” [Online]. Available: http://www.broadband-forum.org. [Accessed 30 6 2014].

[17] XO Communications, LLC, „Managed Firewall and VPN Service Terms & Conditions,“ [Online]. Available: http://www.xo.com/legal-and-privacy/managed-firewall-and-vpn-services-terms-and-conditions/. [Zugriff am 30 06 2014].

[18] Virgin Media Business LTD, „Managed Firewall,“ [Online]. Available: http://www.virginmediabusiness.co.uk/Products-and-solutions/Broadband-and-Internet-Services/Managed-Security-Services/Managed-Firewall/. [Zugriff am 30 06 2014].

[19] Sungard Availability Services, „Managed Firewall & VPN Services,“ [Online]. Available: http://www.sungardas.com/Solutions/managed-security-services/managed-firewall-and-vpn/Pages/firewall-services.aspx. [Zugriff am 06 30 2014].

[20] T. Roscoe, "The end of Internet architecture," vol. In Proceedings Of The Fifth Workshop On Hot Topics In Networks, 2006.

[21] B. Liskov, "The Power of Abstraction," [Online]. Available: http://www.pmg.lcs.mit.edu/~liskov/turing-09-5.pdf. [Accessed 27 06 2014].

[22] J. C. Doyle, J. Carlson, S. H. Low, F. Paganini and G. Vinnicombe, "Robustness and the Internet: Theoretical Foundationa," 5 03 2002. [Online]. Available: http://www.maoz.com/~dmm/complexity_and_the_internet/robustness_and_the_internet_theoretical_foundations.pdf. [Accessed 30 06 2014].

[23] L. Caywood, "Notes from ONS 2014 Day 0: SDN Market Opportunities Tutorial," [Online]. Available: http://theborgqueen.wordpress.com/2014/03/03/notes-from-ons-2014-day-0-sdn-market-opportunities-tutorial/. [Accessed 18 06 2014].

[24] M. Rost and S. Schmid, "VirtuCast: Multicast and Aggregation with In-Network Processing," in *Proceedings of 17th International Conference On Principles Of DIstributed Systems (OPODIS) 2013*, Nice, France, 2013.

[25] ETSI, „Terminology definitions (ETSI GS NFV 003 v1.1.1),“ October 2013.

[26] P. Quinn and T. Nadeau, "Service Function Chaining Problem Statement (draft-ietf-sfc-problem-statement-03.txt)," IETF, http://tools.ietf.org/html/draft-ietf-sfc-problem-statement-03, 1. April 2014.

[27] Y. Jiang and H. Li, "An Architecture of Service Function Chaining (draft-jiang-sfc-arch-01.txt)," IETF, http://tools.ietf.org/html/draft-jiang-sfc-arch-01, 14. February 2014.

[28] M. Boucadair, C. Jacquenet, R. Parker, D. R. Lopez, J. Guichard and C. Pignataro, "Service Function Chaining: Framework & Architecture (draft-boucadair-sfc-framework-02)," IETF, http://tools.ietf.org/html/draft-boucadair-sfc-framework-02, 12. February 2014.

[29] P. Quinn and A. Beliveau, "Service Function Chaining (SFC) Architecture (draft-quinn-sfc-arch-04.txt)," IETF, http://tools.ietf.org/html/draft-quinn-sfc-arch-04, 28. January 2014.

[30] Amdocs, Amdocs, [Online]. Available: http://www.amdocs.com/whitepapers/posters/etomv9.pdf. [Accessed 23 April 2014].

[31] Open Networking Foundation, „Solution Brief: OpenFlow-enabled SDN and Network Functions Virtualization," [Online]. Available: https://www.opennetworking.org/solution-brief-openflow-enabled-sdn-and-network-functions-virtualization. [Zugriff am 30 6 2014].

[32] W. S. Liu, H. Li, O. Huang, M. Boucadair, N. Leymann, Z. Cao, J. Hu and C. Pham, "Service Function Chaining (SFC) Use Cases (draft-liu-sfc-use-cases-04)," IETF, http://tools.ietf.org/html/draft-liu-sfc-use-cases-04, 28 March 2014.

Annex 1 State of the art: ETSI NFV Industrial Study Group

The major focus of ETSI standards is related to NFV architecture. In this context it considers "to enable and exploit the dynamic construction and management of network function graphs or sets, and their relationships regarding their associated data, control, management, dependencies and other attributes" The ETSI NFV group released Terminology definitions [25] in October 2013, together with other documents as part of its NFV work.

ETSI GS NFV 003 v1.1.1	
Network Function (NF)	A functional building block within a network infrastructure, which has well-defined external interfaces and a well-defined functional behaviour. In practical terms, a Network Function is today often a network node or physical appliance.
Network Function Virtualisation Infrastructure Point of Presence (NFVI-PoP)	An N-PoP where a Network Function is or could be deployed as Virtual Network Function (VNF).
Network Operator	A Network Operator is defined as an operator of an electronics communications network or part thereof. An association or organization of such network operators also falls within this category.
Network Point of Presence (N-PoP)	A location where a Network Function is implemented as either a Physical Network Function (PNF) or a Virtual Network Function (VNF).
Network Service	A composition of Network Functions and defined by its functional and behavioural specification. The Network Service contributes to the behaviour of the higher layer service, which is characterized by at least performance, dependability, and security specifications.
Network Service Provider	A type of Service Provider implementing the Network Service.
Network Stability	The ability of a network to maintain steadfastness or to resume its designated state as soon as possible against change, deterioration or displacement by anomaly that does not exceed its design limit.
NF Forwarding Graph	A graph of logical links connecting NF nodes for the purpose of describing traffic flow between these network functions.
NF Set	A collection of NFs with unspecified connectivity between them.

NFV Framework	The totality of all entities, reference points, information models and other constructs defined by the specifications published by the ETSI ISG NFV.
NFV Infrastructure (NFVI)	The NFV-Infrastructure is the totality of all hardware and software components which build up the environment in which VNFs are deployed. The NFV-Infrastructure can span across several locations, i.e. multiple N-PoPs. The network providing connectivity between these locations is regarded to be part of the NFV-Infrastructure.
NFV Orchestrator	The NFV Orchestrator is in charge of the network wide orchestration and management of NFV (infrastructure and software) resources, and realizing NFV service topology on the NFVI. The NFV Orchestrator operates, manages and automates the distributed NFV Infrastructure. The NFV Orchestrator has control and visibility of all VNFs running inside the NFVI. The NFV Orchestrator provides GUI and external NFV-Interfaces to the outside world to interact with the orchestration software. The NFV Orchestrator makes extensive use of the NFV-Operating System to perform cloud operation and automation, thus extends the basic capabilities of the NFV-OS and provides much richer environment.
NFV-Resource (NFV-Res)	NFV-Resources do exist inside the NFV-Infra and can be used by the VNF/VNSF to allow for their proper execution.
Physical Network Function (PNF)	An implementation of a NF via a tightly coupled software and hardware system.
Scaling	Ability to dynamically extend/reduce resources granted to the Virtual Network Function (VNF) as needed. This includes scaling up/down and scaling out/in
Scaling Out/In	The ability to scale by add/remove resource instances (e.g. VM).
Scaling Up/Down	The ability to scale by changing allocated resource, e.g. increase/decrease memory, CPU capacity or storage size.
Service	A component of the portfolio of choices offered by service providers to a user, a functionality offered to a user, as defined in 3GPP TR 21.905 [2]. A user may be an end-customer, a network or some intermediate entity.
Service Continuity	The continuous delivery of service in conformance with service's functional and behavioural specification and SLA requirements, both in the control and data planes, for any initiated transaction or session till its full completion even in the events of intervening exceptions or anomalies, whether scheduled or unscheduled,

	malicious, intentional or unintentional. From an end-user perspective, service continuity implies continuation of ongoing communication sessions with multiple media traversing different network domains (access, aggregation, and core network) or different user equipment.
Service Level Agreement (SLA)	A negotiated agreement between two or more parties, recording a common understanding about the service and/or service behaviour (e.g. availability, performance, service continuity, responsiveness to anomalies, security, serviceability, operation) offered by one party to another, and the measurable target values characterizing the level of services. Note: The scope of the above definition does not include business aspects of the SLA
Service Provider	A Service Provider is defined as a company or organization, making use of an electronics communications network or part thereof to provide a service or services on a commercial basis to third parties.
User Service	A component of the portfolio of choices offered by service providers to a user, a functionality offered to a user.
Virtual Application (VA)	A Virtual Application is the more general term for a piece of software which can be loaded into a Virtual Machine. A VNSF is just one type of VA amongst many others, which may not relate to any VNF (e.g. SW-tools or NFV-Infra-internal applications).
Virtualised Network Function (VNF)	An implementation of an NF that can be deployed on a Network Function Virtualisation Infrastructure (NFVI).
VNF Forwarding Graph (VNF FG)	A NF forwarding graph where at least one node is a VNF.
VNF Set	A collection of VNFs with unspecified connectivity between them.

Annex 2 State of the art: OpenDaylight

OpenDaylight is an Open Source Software project under the auspices of the Linux Foundation. Its goal is furthering the adoption and innovation of (SDN) through the creation of a common industry - supported platform, with a modular, pluggable, and flexible controller at its core.

The controller exposes open northbound APIs which are used by applications. OpenDaylight supports the OSGi framework and bidirectional REST for the northbound API. The OSGi framework is used for applications (service functions, platform services and extensions that will run in the same address space as the controller while the REST (web based) API is used for applications (network apps, orchestration, and services that do not run in the same address space (or even necessarily on the same machine) as the controller. The business logic and algorithms reside in either of these applications. These applications use the controller to gather network intelligence, run algorithms to perform analytics, and then use the controller to orchestrate the new rules.

OpenDaylight can be used as the network controller "plugin" in the OpenStack cloud operating system.

OpenDaylight is a candidate to be used in the UNIFY architecture as a network controller. It's open source with a significant industry momentum. The various south band interfaces (like OpenFlow 1.0, 1.3, OVSDB) could provide compatibility with various networking equipment, including the Universal Node. The high number of possible applications, including the ones providing network overlays, and the abstraction capability of the SAL provide a flexible structure, which is already partially shown in the different releases of the controller.

ODL technologies

The controller is implemented strictly in software and is contained within its own Java Virtual Machine (JVM). As such, it can be deployed on any hardware and operating system platform that supports Java.

Maven is used to handle module dependencies and for building the system.

Open Services Gateway Initiative (OSGi) defines ODL's modular architecture:

- Allows dynamically loading bundles
- Allows registering dependencies and services exported
- Provider means for exchanging information across bundles

Java Interfaces are used for event listening, specifications and forming patterns.

ODL projects, releases and interfaces

OpenDaylight is a multi-project, currently there are 15 projects running. Components from sub-projects are brought together in a simultaneous release. The first release (Code name: Hydrogen) is planned to be released in 2014. There are several "editions" (base, virtualization, service provider), to group related functionalities together.

Cloud support

The virtualization edition will provide OpenStack integration. OpenDaylight exposes a single common OpenStack Service Northbound (see Figure 8.2). The API exposed matches the Neutron API of OpenStack precisely; therefore from OpenStack point of view the ODL's Neutron plugin simply passes through the request, which is handled in ODL's Neutron Service.

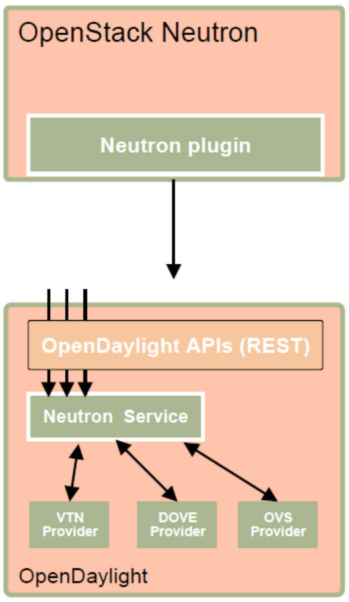


Figure 8.2: OpenStack & OpenDaylight

Annex 3 State of the art: IETF Service Function Chaining (SFC)

Service Function Chaining working group [9] at IETF was approved and established in November 2013 (IETF 88 in Vancouver). It was formerly referred as Network Service Chaining group. The work is at an early stage, however, several drafts have already been submitted focusing on different aspects of service chaining and related technical issues. There are proposals on the architecture as well, but it is worth noting that different branches of drafts are converging.

From architectural aspects, the relevant drafts of the working group are summarized below.

Table 8.1: Relevant IETF drafts

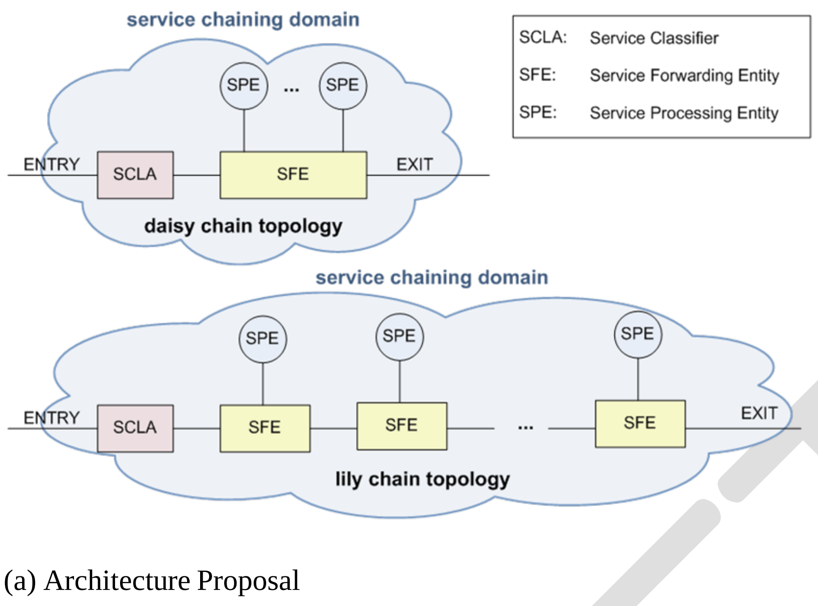
Draft name	Title	Publication Date
http://tools.ietf.org/html/draft-ietf-sfc-problem-statement-03	Service Function Chaining Problem Statement	Apr 1, 2014
http://tools.ietf.org/html/draft-jiang-sfc-arch-01	An Architecture of Service Function Chaining	Feb 14, 2014
http://tools.ietf.org/html/draft-boucadair-sfc-framework-02	Service Function Chaining: Framework & Architecture	Feb 12, 2014
http://tools.ietf.org/html/draft-quinn-sfc-arch-04	Service Function Chaining (SFC) Architecture	Jan 28, 2014
http://tools.ietf.org/html/draft-liu-sfc-use-cases-04	Service Function Chaining (SFC) Use Cases	Mar 28, 2014

Today service deployment, service provisioning and service chaining have several limitations in terms of dynamicity, flexibility, scalability, optimal usage of resources, etc. Current service chaining mechanisms are strongly coupled to the physical network topology and the capabilities of special purpose, expensive HW elements. As a consequence, configuring/deploying service chains is a complex task and usually requires manual processes. The dynamic nature of service chains also produces several challenges. The main problems are highlighted in [26].

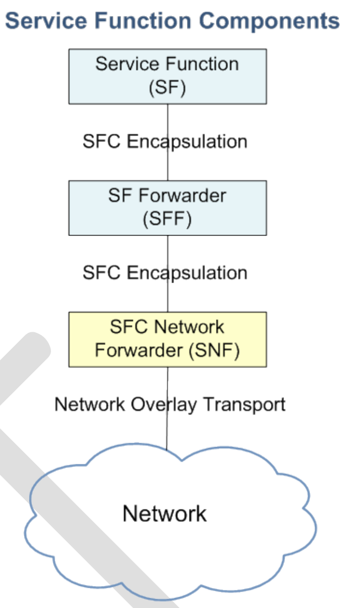
In order to mitigate these problems, an initial abstract architecture for service chaining has been recently proposed by IETF [27]. It is a flexible and scalable architecture which can fulfil requirements of service chaining.

The proposed service chaining architecture is depicted in Figure 8.3 and it consists of the following components:

- Service Classifier (SCLA)
- Service Forwarding Entities (SFEs)
- Service Processing Entities (SPEs)



(a) Architecture Proposal



(b) Components

Figure 8.3: Service Function Chaining architecture and components proposed by IETF

Incoming packets at the entry point are classified into different service flows by SCLA based on policies or flow characteristics. SFE forwards packets to SPEs according to the configuration of service chains. While packet processing tasks (network functions) are implemented by SPEs. Completing all processing tasks, packets leave the service chaining domain at the exit point. SCLA, SFE and SPE can be implemented as standalone blocks or combined modules run by single HW elements or SW modules running in data centres. Different service chaining topologies are supported by this architecture as shown in the figure. In case of a daisy chain topology, multiple SPEs can be attached to single SFE and SPEs cannot send packets directly to each other but only via shared SFEs. When lily chain topology is used, a single SPE is attached to a single SFE and SFEs are connected in a sequence to each other. Hybrid chains can combine the two approaches. A service chaining controller is also necessary to centrally manage and configure service chains (e.g., set up, remove, monitor), which can be implemented together with an SDN controller or a network orchestrator.

The IETF draft [28] proposes a similar framework to enforce Service Function Chaining (SFC) with minimum requirements on the physical topology of the network. The proposed framework allows for differentiated forwarding: packets are initially classified and marked at the entry point of an SFC- enabled domain, and are then forwarded on a per SF Map Index basis. For these purposes, a couple the concepts SF Identifier and SF Map are introduced. The first refers to a unique identifier that unambiguously identifies an SF within an SFC-enabled domain. Consequently, a (single-domain) service chain is defined by an ordered list of SF identifiers, forming the SF map. Packets entering the SF domain at a boundary node are classified according to a SFC Policy Table, linking classified packets to SF maps and sending towards the locator of first SF node of that chain. The document also gives an overview on potential encapsulation headers to store SF Map Indexes of packets once they have been classified at the ingress.

In [29], another architecture is described which can be used for the creation of Service Function Chains (SFC) in case of a single network administrative domain. It includes architectural concepts, principles, and components to provide SFCs in such network environment. The main logical components shown in Figure 8.3 (b) are the following:

The components of the architecture can be defined as follows:

- SF – Service Function: resource described by network locators, variable set of attributes; send/receive SFC encapsulated data to/from one or more SF Forwarders
- SFF – SF Forwarder: service layer forwarding
- SNF – SFC Network Forwarder: forwarding along SFPs based on info contained in SFC encapsulation
- SFP – Service Function Path: the instantiation of SFC
- SFC Encapsulation: enables the sharing of metadata/context
- Overlay Service Topology: formed between SNFs

Annex 4 State of the art: OpenStack

OpenStack is a cloud operating system that controls large pools of compute, storage, and networking resources throughout a data centre.

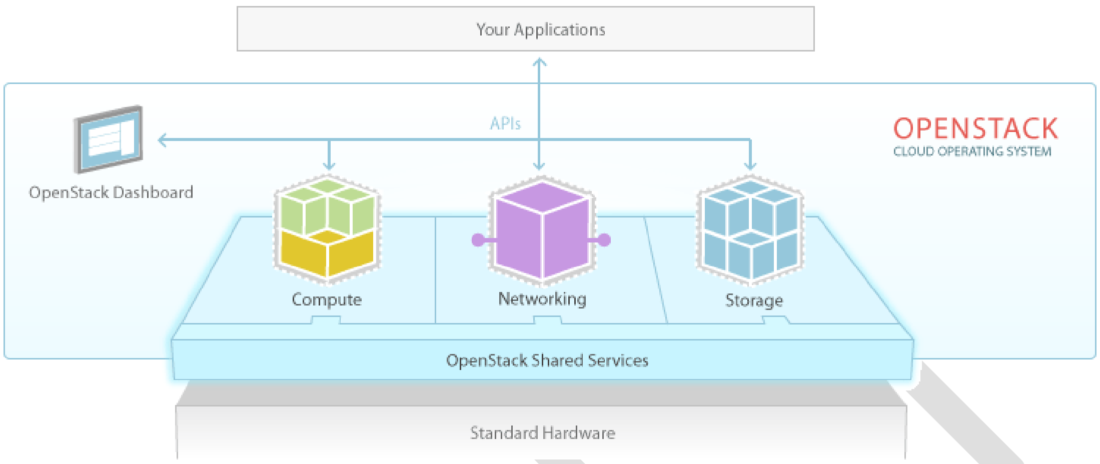


Figure 8.4: OpenStack overview

OpenStack consists of the following main components:

- Compute (codenamed "Nova") provides virtual servers upon demand. There are commercial compute services built on Nova and used internally at companies like NASA (where it originated).
- Network (codenamed "Neutron") provides "network connectivity as a service" between interface devices managed by other OpenStack services (most likely Nova). The service works by allowing users to create their own networks and then attach interfaces to them. OpenStack Network has a pluggable architecture to support many popular networking vendors and technologies, for example OpenDaylight can be used.
- Image (codenamed "Glance") provides a catalogue and repository for virtual disk images. These disk images are mostly commonly used in OpenStack Compute.
- Object Store (codenamed "Swift") provides object storage. It allows you to store or retrieve files (but not mount directories like a fileserver). Several companies provide commercial storage services based on Swift.
- Dashboard (codenamed "Horizon") provides a modular web-based user interface for all the OpenStack services. With this web GUI, one can perform most operations on the cloud like launching an instance, assigning IP addresses and setting access controls.
- Identity (codenamed "Keystone") provides authentication and authorization for all the OpenStack services. It also provides a service catalogue of services within a particular OpenStack cloud.
- Block Storage (codenamed "Cinder") provides persistent block storage to guest VMs.
- Orchestration (codenamed “Heat”) implements an orchestration engine to launch multiple composite cloud applications based on templates.

The basic relations between the components are depicted in the following figure:

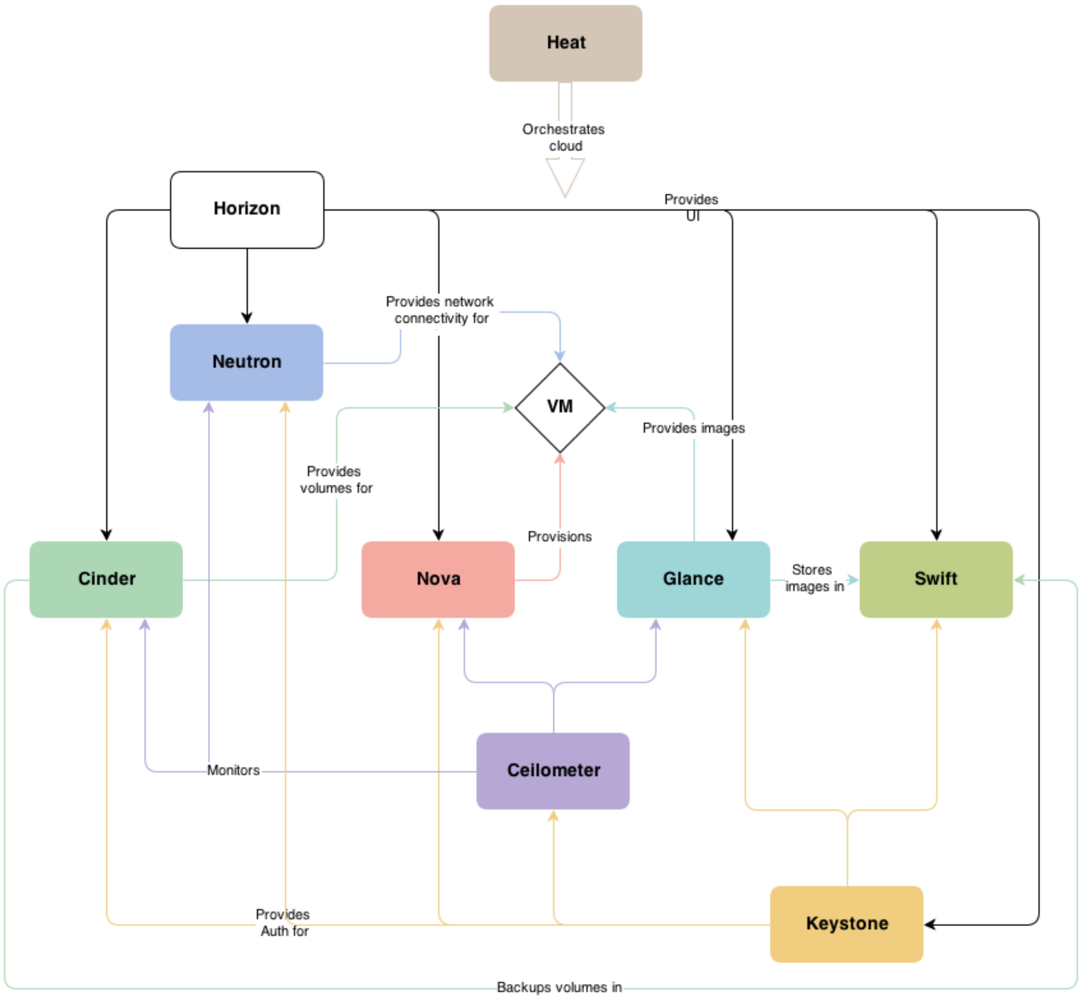


Figure 8.5: OpenStack components

Each of the components provides an API to access it, which can be used as a REST API with the http protocol, or as CLI [13]. OpenStack, as an open-source data centre implementation is a candidate to be managed and orchestrated by UNIFY. The open codebase and interfaces, the widespread use by industry and the wide selection of pluggable hypervisors and networking components supports to use it in UNIFY.

The computation (Nova) API is a candidate to be used for controlling the computing resources in the Unified Node. The networking (Neutron) interface used for intra-data centre networking provides a subset of the networking functionality needed for the whole UNIFY scope. The orchestration (Heat) interface shows a subset of cloud application parameters, to be considered when defining UNIFY Service Graphs (different definition of orchestration than UNIFY).

Annex 5 State of the art and analysis: TMForum/eTOM processes

The TMForum has defined a complete suite covering the management and business aspect of operating networks called Frameworx available in version 13.5 [14]. It is split into four major parts:

- Business Process Framework (eTOM) is a hierarchy of typical business processes and requirements covered in an enhanced telecommunications operations map (eTOM) with a special focus on running an enterprise
- Information Framework (SID) provides a reference for implementing the processes as "off-the shelf information " with a shared information and data model (SID)
- Application Framework (TAM) provides a list for covering the practical, widely adopted tools for fulfilling the processes called telecom applications map (TAM)
- Integration Framework - architecture and standard interfaces provides a guideline how to combine the application framework with the business process and the information framework

In principle the first three parts cover the same aspects but from different viewpoints, the latter one integrates the three views into one overarching concept. Each of the elements provides high-level aspects and iterates via several levels into more details.

TMForum's history resides in the telecommunication network area and it lately adopted IT-related aspects as well as latest technology trends. This is reflected in new work groups like the ZOOM project (zero-touch orchestration, operations and management) or the active SDN/NFV community inside TMForum.

UNIFY targets specifically carrier networks and could borrow some of the ideas, basic principles and information, which are proofed to be a good basis for understanding and covering operational aspects in the design of the architecture.

For the upcoming developments the following aspects of TMForum should be taken into account as well:

- Sequential process from basic to fine granular levels
- Use of process map and process descriptions to design and verify architecture
- Select only what is useful, skip or redesign any unnecessary aspect

More specifically, the Business Process Framework is most relevant in the beginning. As said, the framework is based on a multi-layer decomposition process; there from a general process more fine-granular processes and parts are identified. Each of the processes is described in detail, but one does not have to follow all details and can adapt to individual needs. The Business Process Framework consists of three major parts:

- Strategy, Infrastructure & Products which covers more planning and general management activities
- Enterprise management which covers the supporting activities financial, human resources or risk management

- Operations area which defines the core activities of an enterprise and describes all the daily operational aspects

It is obvious that the primary focus of UNIFY is on the operations area. The other areas seem to have lower relevance for UNIFY as the focus is neither on the operation of an enterprise as such nor on the detailed general and strategic planning issue, which will be most likely required for a real deployment only.

The analysis will start at this level in the eTOM process framework in order to identify the impact of the basic process areas and select the most important aspects. In the following, it is not claimed to fully represent eTOM or to model all processes in detail: a pragmatic analysis with respect to the architecture design of UNIFY is performed. Second, the missing part of the IT-based DevOps concept will be analysed and a general, converged process framework for the UNIFY project defined.

The analysis of part one starts with the overview of the eTOM process map as depicted in Figure 8.6. UNIFY is not into enterprise management as such and has some limited connections to Strategy, Infrastructure & Products. Here the important areas are Service and Resource Development and Management in the Infrastructure and Product Lifecycle Management. The connections are mainly with

- Product lifecycle management with the gathering and analysis of resource service requirements
- Enablement of service and resource development and deployment
- Development of service and resource specifications
- Termination (exit) of resources and services.

In practice, this area is dedicated to management activities and UNIFY will advance here with automation and the SP DevOps concept. To a large extent other processes which commit management activities are part of this block as well (e.g. management of research or business plan production), but they are not dedicated to run the infrastructure and service environment and are therefore left out.

The major conclusion is to commit to the ability to deploy, extend and release infrastructure and services in the UNIFY environment, something which is anyhow considered to be important for a complete architecture and process framework.

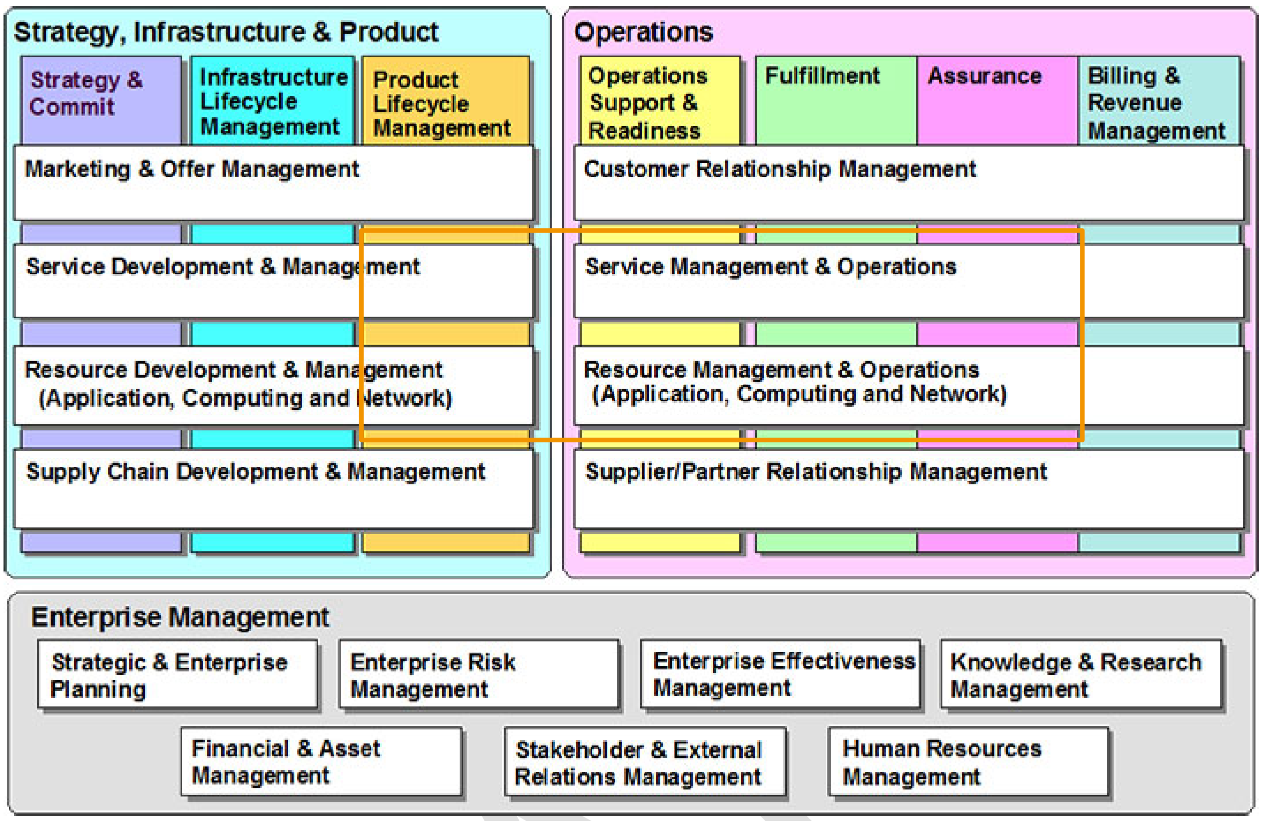

Strategy, Infrastructure & Product
Strategy & Commit
Infrastructure Lifecycle Management
Product Lifecycle Management
Marketing & Offer Management
Service Development & Management
Resource Development & Management (Application, Computing and Network)
Supply Chain Development & Management
Operations
Operations Support & Readiness
Fulfillment
Assurance
Billing & Revenue Management
Customer Relationship Management
Service Management & Operations
Resource Management & Operations (Application, Computing and Network)
Supplier/Partner Relationship Management
Enterprise Management
Strategic & Enterprise Planning
Enterprise Risk Management
Enterprise Effectiveness Management
Knowledge & Research Management
Financial & Asset Management
Stakeholder & External Relations Management
Human Resources Management

Figure 8.6: eTOM process map and selected hot spots

The most important area is "Operations", which itself splits into a matrix of four x four:

- Horizontal processes:
  - Customer Relationship Management (CRM) deals with customer interfaces, the ordering process and any problem handling and billing
  - Service Management & Operations (SM&O) processes include the deployment, installation, configuration, management, analysis as well as providing billing information. Quality aspects of the service are covered here as well
  - Resource Management & Operations (RM&O) is similar to the previous aspects on service, but with focus on the infrastructure and its elements like network equipment or IT software components
  - Supplier/Partner Relationship Management (S/PRM) deals with all interactions to external partner. It provides interfaces for ensuring end-to-end delivery
- Vertical processes:
  - Operations Support and Readiness deals with all aspects needed to setup and to run environment. It is supposed to be a less "real-time" oriented process like the other three areas. Examples include self-provisioning portals, resource description transmissions as well as auto-configurations

- Fulfilment process provides customers with their requested services. Here the needs and the quality expectations are translated and especially network and service environments are configured
- Assurance process is responsible for maintaining and monitoring the environment. This includes monitoring of information as well as analysis and generation of alarms, notifications etc.
- Billing process collects the usage information and translates it into billing records.

UNIFY is active in many of these processes. In order to focus on the core aspects, the following aspects are excluded:

- CRM is acknowledged by assuming that customer input is integrated in one or the other form (e.g. web portal, BSS information, etc.), but it is not desired to do specific developments in UNIFY. In addition, certain information like customer locations, QoE/SLA requirements as well as management will be captured on a high level only
- S/PRM is not an individual area of interest for UNIFY. Selection, assessment and monitoring on a per supplier/partner basis is not target, but the focus is on individual (abstracted or virtualised) resources and multi-provider will be a key feature of the architecture in any case with some layered approach, potentially with some degree of recursiveness. So lower layers are always seen as being the own, interfaced layer with the small difference of being external. An open issue is the management of the access in terms of authorisation and authentication which is basically and in current state out of scope of UNIFY
- Billing process is left out, it is merely a summary of resource usage concatenated with price information, again out of scope for UNIFY

To conclude, the most interesting area of the eTOM process chart for UNIFY, is marked with an orange rectangle in Figure 8.6 surrounding Operations Support and Readiness, Fulfilment, Assurance as well as Service and Resource Management and Operations.

Annex 6 Use Cases

UC1: Re-use of existing fixed network resources by Mobile Network Operator

Motivation: Exploitation of SDN and NFV to accommodate easy deployment of mobile networks, and thus, accelerate the roll-out of mobile broadband.

Description: In this use case, the (Fixed) Network Operator employs infrastructure virtualization in order to enable dynamic sharing of his resources in backhaul links and routers, so as to handle traffic generated by both fixed and mobile access networks. Note that the mobile access network may belong to a different business entity, e.g. a Mobile Network Operator (MNO), thus backhauling of the mobile traffic is offered as a wholesale service to the MNO by the Network Operator. The virtualization of the infrastructure in the core network of the Network Operator will enable a flexible and dynamic deployment and provisioning of cloud services and applications to the rapidly increasing number of mobile users. Moreover, it will allow the development of new services and applications, and thus, lower the technical barrier to share access/ aggregation infrastructure in-between operators and enabling business with application service providers and content owners.

Characteristics: Major characteristics of UC1 include the high mobility of users, limited energy storage (battery) in users' device, main functionality of applications and services moved into the cloud, while only a shell providing access to the service is running in the mobile device.

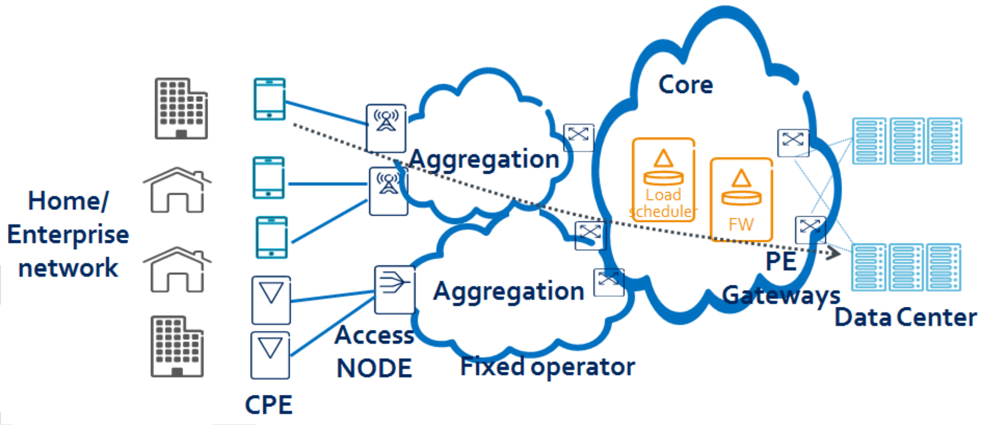


Figure 8.7: The traffic of the mobile access network is handled by the virtualized infrastructure of the fixed Network Operator.

UC2: Interoperability in multi-operator environments – share of NSC functions across different infrastructure/network domains

Motivation: Efficient handling of traffic crossing multiple domains.

Description: In this complex case, the end-users may experience heavy Quality of Experience (QoE) deterioration in terms of delay depending on the cross-domain network load and the number of the policy elements through which data is passing. This UCG can be beneficial to the operators, providers and to the end users since through sharing/consolidating physical resources among various operators or abstracting some network functional operations to the upper layers rough SDN, it can lead to CAPEX and OPEX reductions for the operators and to the savings for the end user. With dynamic NSC implemented in each of the network domains and by exploiting the NSC functionality mainly at the edge gateways, it will lead to more intelligent traffic steering and thus provide traffic performance acceleration.

Characteristics: The major characteristic of UC2 is the consideration of a multi-provider environment and the business agreements between the different domains/providers, e.g. peering or transit agreements, as well as revenue sharing issues out of the cloud services delivered.

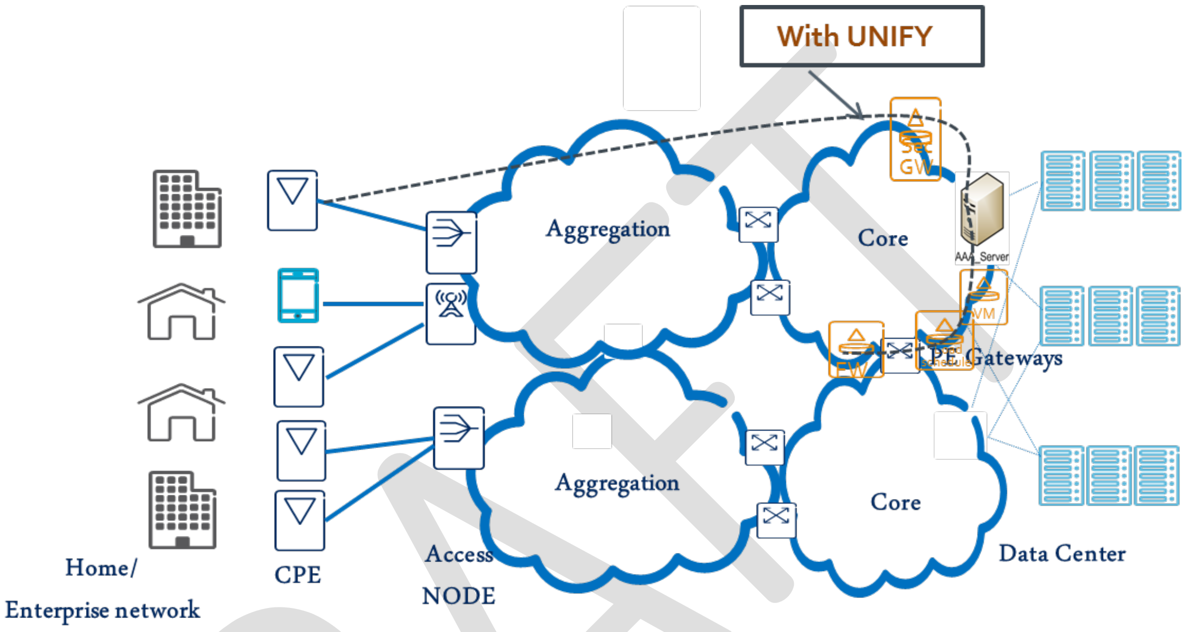


Figure 8.8: Interoperability in multi-operator environments - share of NSC functions across different infrastructure/network domains.

UC3: Multi Access Technology Support

Motivation: Aggregation of available resources of multiple, different service creations for a service / user / device by supporting mobility and ad-hoc connectivity (e.g., WiFi AP).

Description: In this use-case, a user wants to combine his fixed and mobile connection in order to leverage the combined, overall capacity (i.e., bandwidth) with the help of a load balancer. The different fixed and mobile service creation (including from other operators like public WiFi AP) might have different requirements per chain like enhanced security (requiring dedicated GW) or different elements like NAT, parental control, policy elements, etc. All elements could be virtualized and therefore flexibly distributed over the different network elements / processing facilities, e.g. Data Center.

Characteristics: The major characteristic of UC3 is the consideration of multiple access technologies in the end-user side, e.g. xDSL, WiFi, LTE, etc. and the integration/convergence of the management of traffic generated by each different interface in a dynamic and flexible manner.

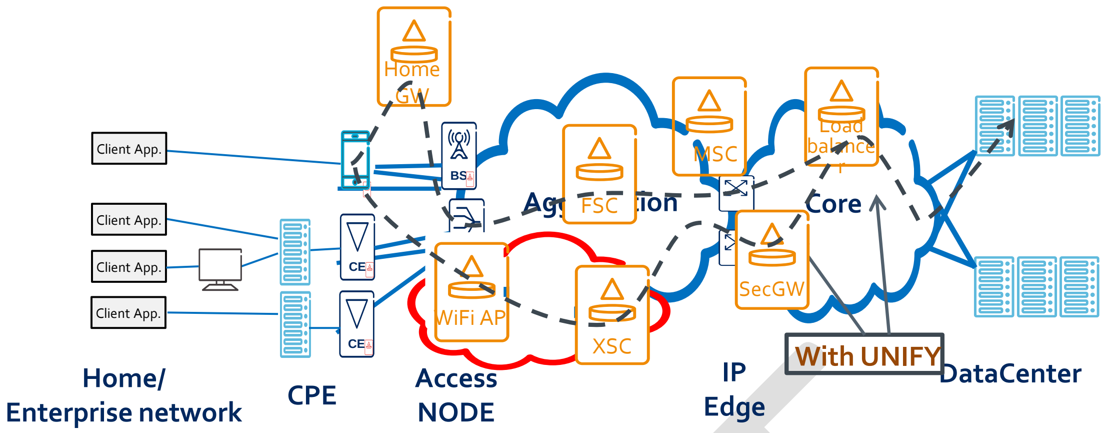


Figure 8.9: Support of multi-homing while employing different access technologies.

UC4: Fulfil rapidly evolving and heterogeneous service demand

Motivation: Flexible and cost-effective operation and maintenance of the network, while providing integrated cloud services with improved QoS/QoE.

Description: In this use case, UNIFY will enable the dynamic sharing of capacity in both aggregation and core links between traffic flows generated by different cloud services and mainly with different traffic characteristics and QoS/QoE requirements. Moreover, it will enable flexible and efficient deployment and provisioning of increasingly popular cloud services such as storage and computation, as well as applications (e.g., video streaming, online storage, and gaming).

Characteristics: The major characteristic of UC4 is the diversity of cloud services and applications, and the associated QoS/QoE requirements.

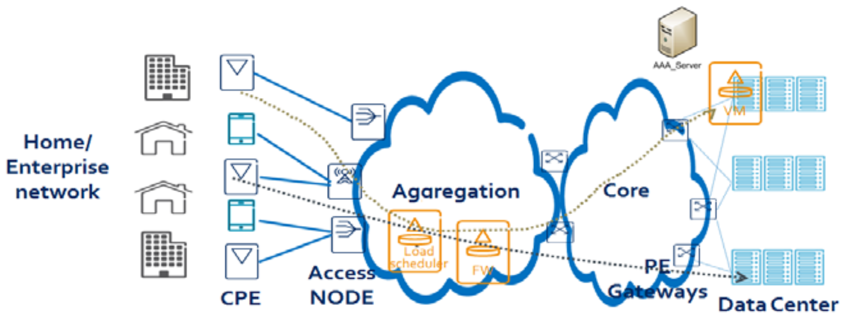


Figure 8.10: Fulfilment of rapidly evolving and heterogeneous service demand.

UC5: Virtual Residential Gateway – Border Network Gateway (BNG)

Motivation: Flexible management of a multitude of customized BNG functions and configurations, which are employed due to device/user-aware services that allow fine-grained control of customer devices, as well as due the support of roaming users.

Description: In this use case, UNIFY will provide service abstraction and virtualization of the BNG functionality. The virtualization of BNG will enable the separation and migration of some of its services / functionalities, e.g. load balancing, firewall, to the network. The migration of BNG services in the network and the service chaining of different functionalities w.r.t. to the access technology (wired or wireless), the users' profiles, and the QoS requirements associated with the cloud services that each user enjoys, will allow a dynamic and highly flexible customization of the BNG individually per customer.

Characteristics: Major characteristics of UC5 comprise the development and provision of new services and applications offered to end-users, and the reduction of CAPEX and OPEX by sharing resources between the Network Operator and the end-user.

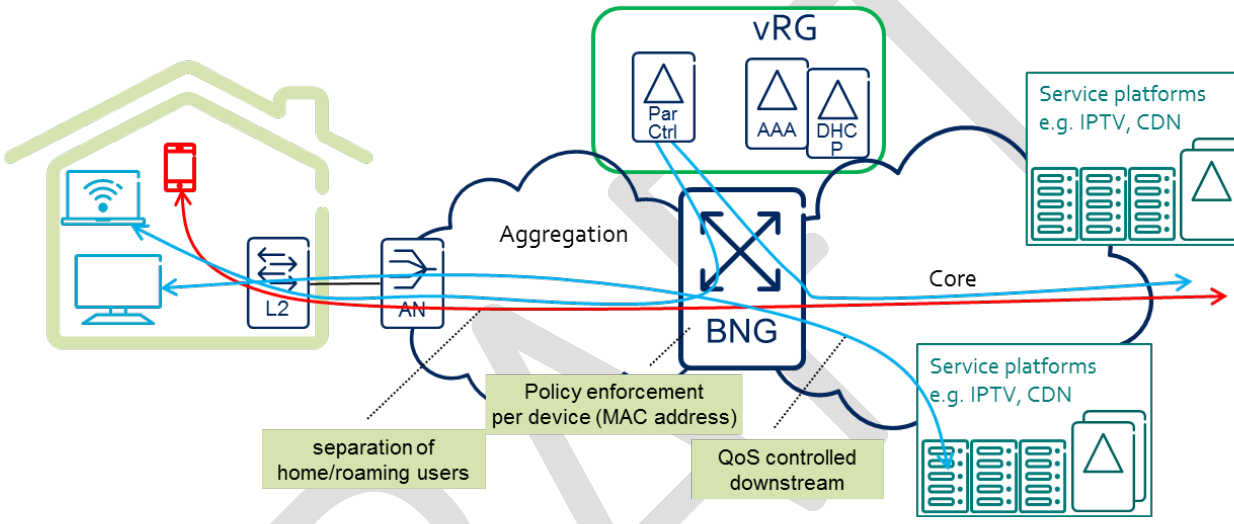

vRG
Par Ctrl
AAA
DHCP
Service platforms e.g. IPTV, CDN
Aggregation
Core
L2
AN
BNG
Service platforms e.g. IPTV, CDN
Policy enforcement per device (MAC address)
separation of home/roaming users
QoS controlled downstream

Figure 8.11: Virtualization of Border Network Gateway (BNG)

UC6: Fast and Flexible Provisioning of Value-Added Services from the Operator

Motivation: Reduction of time-to-market for the introduction of new value-added services; move from today's limited, fixed and static service profiles to fully flexible and customized provisioning of service features (e.g., DPI, firewall, URL filtering, proxy/NAT, caching) to individual users; reduce the operational complexity involved.

Description: The use case aims at showing the flexibility in the SDN-controlled delivery of network functions to the users and the role of cloud based virtualization. A flexible subset of the above network functions has to be dynamically associated to a user's traffic, even on a per-application basis, with minimal operational complexity. A few evolutionary steps can be as follows:

- A set of diverse network functions (NF1, NF2, etc.) is available in the POP, which are initially implemented by dedicated physical network appliances. The network functions to be considered for the use case may consist in the following: DPI, firewall, URL filtering, proxy/NAT and caching.
- Some of the above network functions migrate in the form of virtual appliances, i.e., they are now implemented as virtualized network functions (VNF) on a server hypervisor environment in the POP.

- Some of the virtualized network functions can be moved even more “far away” in a centralized cloud data centre.

Characteristics: Major characteristics of UC6 include the features and QoS requirements of the individual value-added services, the virtualization of the various network functions related to the Edge POP, and the migration of some of those in the network or cloud domain.

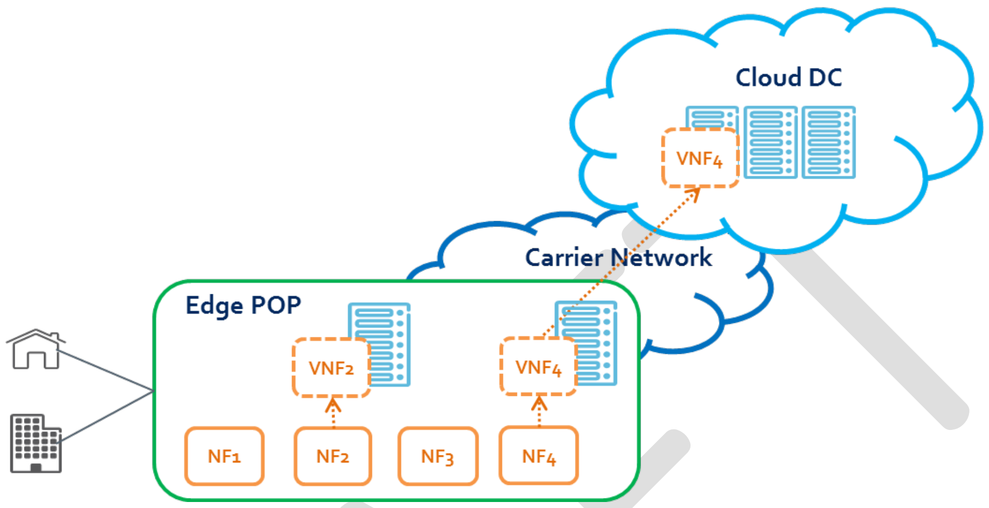


Figure 8.12: Provisioning of value-added service by means of network function virtualization and migration in the network or cloud domain.

UC7: User-provisioned services on network edges

Motivation: Reduce time- to-market for the introduction of new value added services; move from today’s limited, fixed and static service profiles to fully flexible and customized provisioning of service features (e.g. DPI, firewall, URL filtering, Proxy/NAT, caching) to individual users; reduce the operational complexity involved.

Description: A flexible subset of the above network functions has to be dynamically associated to a user’s traffic, even on a per-application basis, with minimal operational complexity.

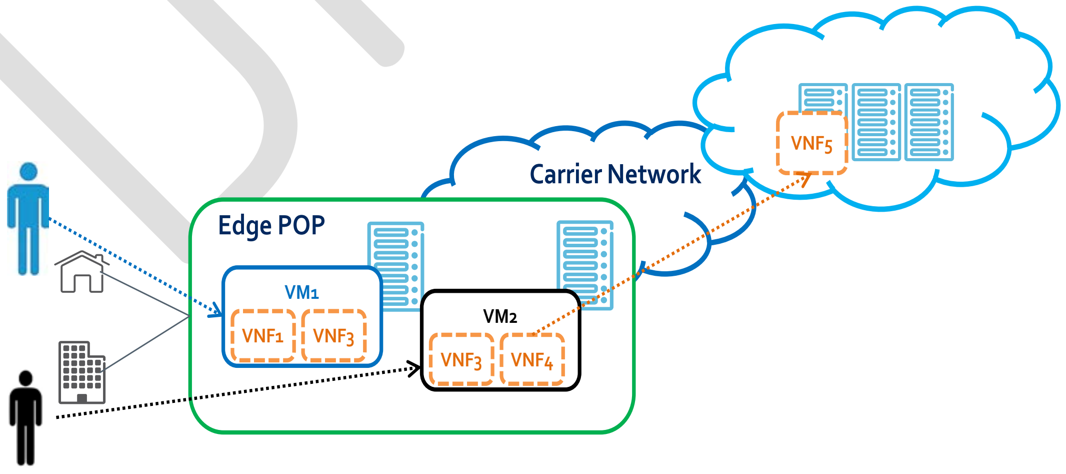


Figure 8.13: User-provisioned services on network edge nodes

UC8: 3D Life logging use case description

Motivation: The use case considers a service distribution to a number of users. The service chain is instantiated per-provider (meaning that there is one instance of the service that handles all the users). In practice, different links in the service chain will be implemented in a multi-threaded manner or use distributed programming techniques. Although the chain is instantiated once, different links in the chain are activated differently for each user.

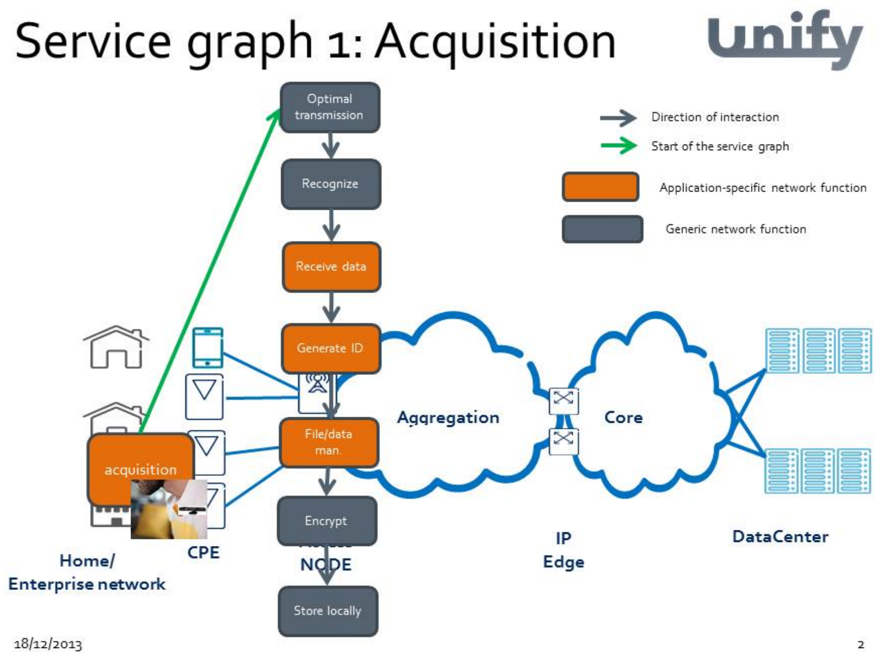


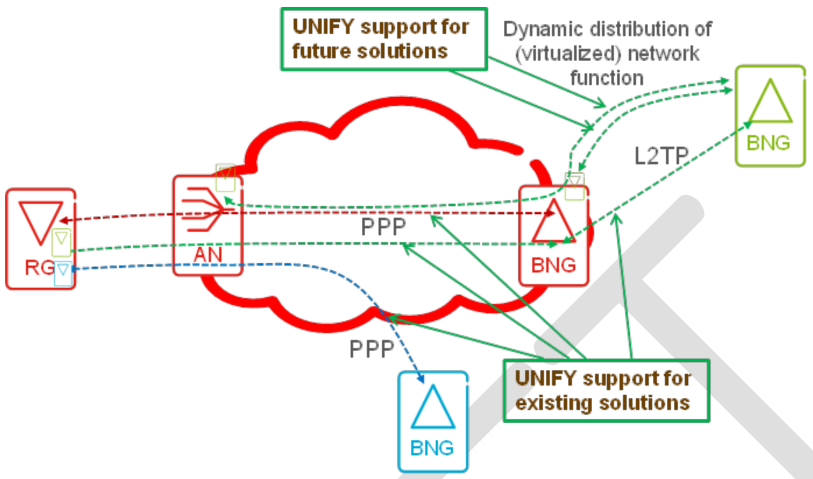
Figure 8.14: Video acquisition and local reverse caching

UC9: Wholesale Operator – Network Service Chain as a Service (NSCaS)

Motivation: Retail Operator to flexibly configure network and service function resources from the Wholesale Operator.

Description: A Retail Operator aims to configure the subscription of his end-user, which results in update of the configuration in the network of the Wholesale Operator. The configuration may include adding/removing subscriber, modifying/upgrading/downgrading service, deployment of added-value services such as IPTV, which requires non-automated (manual) and off-line configuration of Access Control Lists (ACL) in the premises of the Wholesale Operator. Moreover, the propagation of any change performed at the edge of the network to the core of it implies increase of operational expenses for both Operators.

Virtualization of the associated network functions will allow a dynamic configuration of users' profiles, deployment of new services, etc.; thus, it will enable a flexible and more efficient provision and management of network services by the Wholesale Network Operator.



UNIFY support for future solutions
Dynamic distribution of (virtualized) network function
BNG
L2TP
RG
AN
PPP
BNG
PPP
UNIFY support for existing solutions
BNG

Figure 8.15: Support of efficient and flexible wholesale of network services. (Green and Light Blue entities are retail operators, while Red entity is a wholesale operator.)

UC10: New Service Provider Activation

Motivation: Lower technological and business entry level barriers for new Service Providers.

Description: This use case studies the complete procedure of a new Service Provider accessing the resources exposed by the Network Operators to deploy a new service. This procedure and the interfaces between both entities, i.e., Service Provider and Network Operator, should be clearly defined. The functions covered by each of these providers should be also separated (e.g. which one performs the optimization and under which criteria).

This scenario also highlights the multi-service and multi-provider/operator dimension behind the concept of the UNIFY project. At least, the two aforementioned roles are clearly identified. Moreover, a multi-Network Operator setup can be explored, where they share the same physical resources; in this case, a new role needs to be played, the Infrastructure Provider.

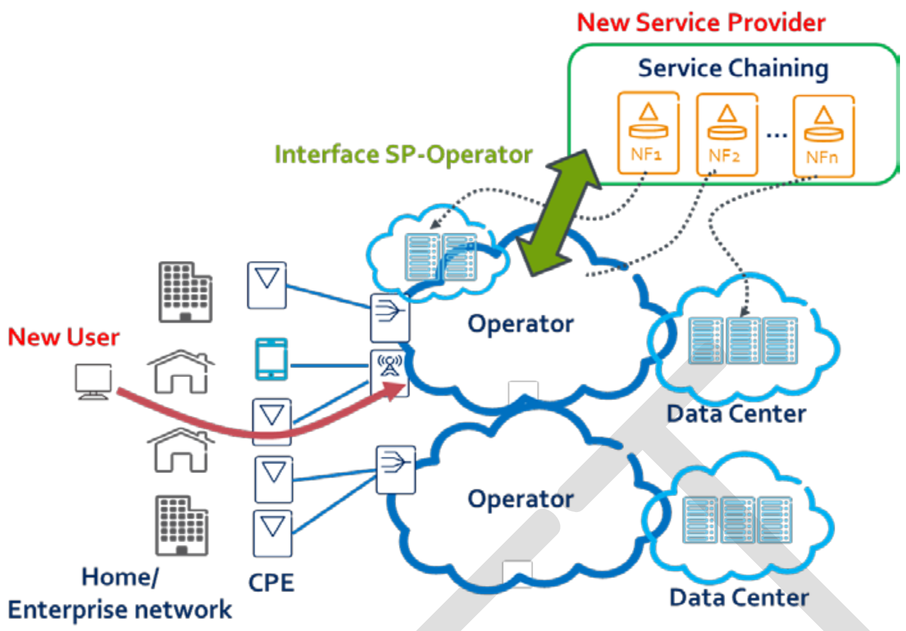


Figure 8.16: Easy entry of new Service Provider into the market.

UC11: QoS support for Over-The-Top Providers

Motivation: Over-The-Top (OTT) providers typically only see best-effort Internet; therefore, it is hard for them to implement delay- or bandwidth-sensitive applications, such as video streaming or voice call.

Description: In this UC, an OTT provider requests a specific class of service is assumed, e.g. a low delay one, or some bandwidth guarantee for its application. The QoS provisioning could be ensured only within the domain of the Network Operator, end-to-end from the OTT premises to the end-users' premises. Of course, some form of compensation should be provided by the OTT to the Network Operator for providing him better than best-effort services; thus, accounting, charging and billing must accompany the QoS service offered to the OTT. Today this can be done with DPI and bearer configuration – only for mobile UEs, which makes it expensive in terms of processing resources and limited in terms of terminals.

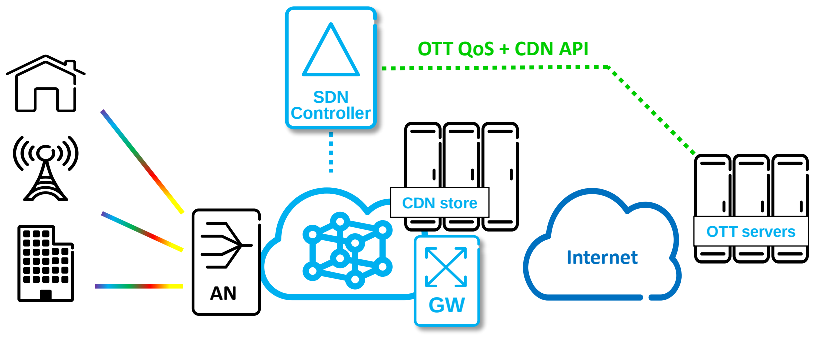


Figure 8.17: Flexible QoS provisioning for OTTs by means of service chaining