Deliverable 2.2: Final Architecture

Dissemination level	PU
Version	1.0
Due date	31.10.2014
Version date	15.11.2014

This project is co-funded
by the European Union

Document information

Editors

Róbert Szabó (ETH), Balázs Sonkoly (BME) and Mario Kind (DTAG)

Contributors

George Agapiou (OTE), Katia Colucci (TI), Jokin Garay (EHU), Eduardo Jacob (EHU), David Jocha (ETH), Juhoon Kim (DTAG), Antonio Manzalini (TI), Catalin Meirosu (EAB), Felician Nemeth (BME), Kostas Pentikousis (EICT), Fulvio Risso (Polito), Matthias Rost (TUB), Pontus Sköldström (ACREO), Rebecca Steinert (SICS), Tobias Steinicke (TP), Wouter Tavernier (IMINDS), David Verbeiren (INTEL), Vinicio Vercellone (TI), Fritz-Joachim Westphal (DTAG) and Hagen Woesner (BISDN).

Coordinator

Dr. András Császár
Ericsson Magyarország Kommunikációs Rendszerek Kft. (ETH)
Könyves Kálmán körút 11/B épület
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

Executive Summary

This deliverable is a sequel to the initial architecture and use-case definitions that have been documented in [D2.1]. In this deliverable the final architecture for unifying carrier and cloud resources is defined.

Methods of the initial architecture design followed state of the art review, definition of use-cases, requirements and identification of key design principles as documented in [D2.1]. This document, however, focuses solely on the definition of the architecture details.

It was identified that with combined abstraction of compute, storage and network resources one can logically centralize, automate and recursively apply resource orchestrations across domains, technologies, vendors etc. The UNIFY architecture implements such a combined abstractions of resources and allows the overarching optimization. Thus, the UNIFY architecture enables automated and recursive resource orchestration and operation with domain virtualization similar to the recursive network-only virtualization of the Open Networking Forum (ONF) Software Defined Networking (SDN) architecture but also for European Telecommunications Standards Institute (ETSI) Network Function Virtualization (NFV) services. The defined architecture also considers the demands of a Service Provider DevOps (SP-DevOps) regime. Along the monitoring, Verification and Troubleshooting needs of operation in carrier environments, SP-DevOps includes support for Network Function (NF) development. The applied virtualization and orchestration concept is independent of resource or domain size, technology, and hence works from a single node, e.g., the Universal Node (UN) concept, to complete multi technology carrier environments. Moreover the logical centralization of joint compute and network resource orchestration enables direct control and elastic scaling of resources for the deployed NFs.

In the UNIFY architecture:

- three layers (service, orchestration and infrastructure) and a set of reference points have been defined;
- a general information model describing the most important reference points has been identified;
- a Network Function Forwarding Graph (NF-FG) for programming resource orchestration at compute, storage and network abstraction, in accordance with the virtualization, monitoring functions and quality indicators for rapid and flexible service creation has been defined;
- a programmable interface enabling a control and data plane split for network functions and dynamically control of their dedicated resources and management actions has been defined;
- a monitoring framework to complement the quasi static virtualization views for fine granular observability of both virtualized infrastructures and NF-FG-based services has been defined;
- a model-based service decomposition in order to be able to re-use and build services out of elementary (or atomic) blocks has been defined;
- a definition and a frame how $i)$ service programming, orchestration and optimization, $ii)$ service provider DevOps and $iii)$ commodity hardware based networking as well as execution environment can form a unified production environment have been defined;
- a detailed functional architecture has been defined, covering all aspects framed in the overarching architecture including a description of the primitives at the reference points.

Overall, the UNIFY design creates a unified production environment for rapid and flexible service creation through joint resource virtualization and orchestration. While the ambition is similar to ETSI NFV, we believe that it is worth

taking a different architecture approach by generalising ONF SDN principles. In this way, multi-level recursion and better resource control of any NF which has split data- and control-plane promise to be benefits. Prototyping and experimentation in both ETSI and UNIFY will foster our understanding of practical implications of the two different architecture approaches.

The information provided in this deliverable and the previously documented initial version of the architecture [D2.1] cover all essential aspects of the UNIFY architecture. However, the on-going work and achieved results of the technical work packages will detail and verify individual aspects of this architecture.

Contents

List of Figures

1 Introduction

To a large degree there is agreement in the network research, practitioner, and standardization communities that rigid network control limits the flexibility and manageability of speedy service creation; see [Joh+13] and the references therein. For instance, it is not unusual that an average service creation time cycle exceeds 90 hours, whereas given the recent advances in virtualization and cloudification one would be interested in service creation times in the order of minutes [5GP13] if not seconds.

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. European Telecommunications Standards Institute (ETSI) Network Function Virtualization (NFV) [ETS13b] and Open Networking Forum (ONF) Software Defined Networking (SDN) [ONF14] seem to be key technology enablers in the direction of meeting requirements such as cost reductions, service flexibility and new business models.

SDN targets at breaking the vertical integration of network data and control planes to introduce (logically centralized) control plane programmability for novel networking virtualization (abstraction), simplified network (re)configuration and policy enforcement.

NFV targets at virtualizing servers and appliances that provide network functions. One of NFV's value propositions is Capital Expenditure (CAPEX) optimization with the usage of Commodity Off the Shelf (COTS) hardware. This, together with increased operational efficiency is expected to also reduce Operational Expenditure (OPEX). Operational efficiency is achieved by the automation of commission, configuration, resource management, etc. for even an order of magnitude higher number of managed elements than before.

In order to remedy today's limits in flexibility of service creation, the UNIFY project pursues full network and service virtualization to enable rich and flexible services and operational efficiency. We research, develop and evaluate means to orchestrate, verify, observe and deliver end-to-end services from home and enterprise networks through aggregation and core networks to data centers. While focusing on enablers of this unified production environment, we

- develop an automated, dynamic service creation platform, leveraging software defined networking technologies;
- create a service abstraction model and a service creation language, enabling dynamic and automatic placement of networking, computing and storage components across the unified infrastructure;
- develop an orchestrator with novel optimization algorithms to ensure optimal placement of elementary service components across the unified infrastructure;
- research new management technologies (Service Provider DevOps (SP-DevOps)) and operation schemes to address the dynamicity and agility of new services;
- and evaluate the applicability of a universal network node based on commodity hardware to support both network functions and traditional data center workloads.

Throughout the definition of the UNIFY architecture we followed the concepts of layering, abstractions and the definition of selected processes. The discussions of these design concepts and the corresponding processes are documented in Sec. 5 and Sec. 6 of [D2.1].

1.1 Scope

This document presents a revision of the D2.1: "Use Cases and Initial Architecture" deliverable [D2.1]. We reuse and revise D2.1 sections according to the following:

- D2.1 Sec. 1 "Introduction" is relevant.
- D2.1 Sec. 2 "State of the art and related work in progress" is extended with a detailed analysis and comparison of the UNIFY architecture and selected services in Sec. 6.
- D2.1 Sec. 3 "Use Cases" is relevant.
- D2.1 Sec. 4 "Requirements" is relevant.
- D2.1 Sec. 5 "Design principles and general properties of the UNIFY architecture" is relevant.
- D2.1 Sec. 6 "Overarching Architecture" is revised by Sec. 3.
- D2.1 Sec. 7 "Functional Architecture" is revised by Sec. 4.

Regarding the objectives of the Description of Work (DoW), we described and analyzed our use-cases in D2.1. Our reference architecture is presented in this current document, with main components (see Sec. 3.2), abstract interfaces and primitives (see Sec. 4.1), which reflect our priorities of design, functional and business requirements. In order to create a resilient, trustworthy and energy-efficient architecture supporting the complete range of services and applications, we support the full flexibility of NFV, we offer isolation among service request (see Sec. 3.2.7), we enforce policy per service consumer (see Sec. 3.2.8), we introduced embedded monitoring services (see Sec. 3.2.9) to provide enablers for Quality of Service (QoS) support and service resilience at any level of the virtualization hierarchy.

We developed and demonstrated a number of proof of concept prototypes:

- In [Cso+14b] we showed that an Extensible Service ChAin Prototyping Environment (ESCAPE) can be built using Mininet, Click, NETCONF and POX components, corresponding to some of the high level components of the Unify architecture.
- In [Ris+14] we presented that user-specific network service functions can be deployed in an SDN-enabled network node, which are the early steps towards the UNIFY's Universal Node (UN).
- In [Cso+14a] we demonstrated aspects of a multi-level service orchestration in a multi-domain network environment, which shows how an existing data center can be connected to an ESCAPE based environment and projects the implementation of recursive UNIFY interfaces.

We discuss the integration of the prototypes in the perspective of a system architecture (see Sec. 5).

We have also identified the following items related to the architecture definition, which we need to follow-up in the continuation of the project:

resiliency In our approach we logically centralize the main components of our architecture and left resiliency for later investigation together with the design of the integrated prototype.

energy efficiency As mentioned in the previous point, we consider a logically centralized architecture, which might need distributed realization due to resiliency or scalability. These will affect the energy efficiency too. We intend to approach this question once we can measure and learn from our prototypes.

1.2 Document structure

Sec. 2 lists abbreviations, definitions and conventions used in the document. Sec. 3 defines the overarching architecture with an overview, definition of main components and main features and benefits. Sec. 4 dives into the details of our reference points and related further components. Sec. 5 gives insight into the integration work based on the ongoing proof of concept prototyping activities. Sec. 6 compares UNIFY to NFV and SDN architectures and services. Finally, we draw conclusions in Sec. 7.

2 Abbreviations, Definitions and Conventions

2.1 Abbreviations

API Application Programming Interface

BiS-BiS Big Switch with Big Software

BNG Broadband Network Gateway

BSS Business Support System

CA Controller Adapter

CAPEX Capital Expenditure

CAS Controller Adaptation Sublayer

CN Compute Node

COTS Commodity Off the Shelf

DC Data Center

DHCP Dynamic Host Configuration Protocol

DoV Domain Virtualizer

DPDK Data Plane Development Kit

DPI Deep Packet Inspection

DRDB Domain Resource Database

EM Element Management

EMS Element Management Systems

ESCAPE Extensible Service ChAin Prototyping Environment using Mininet, Click, NETCONF and POX

ETSI European Telecommunications Standards Institute

FE Forwarding Element

FW Firewall

GUI Graphical User Interface

HW hardware

IDS Intrusion Detection System

IL Infrastructure Layer

ISG Industry Specification Group

IT Information Technologies

KPI Key Performance Indicator

KQI Key Quality Indicator

LB Load Balancer

LSI Logical Switch Instance

MANO Management and Orchestration

MF Monitoring Function

NAT Network Address Translation

NE Network Element

NF Network Function

NF-FG Network Function Forwarding Graph

NF-IB Network Function Information Base

NFS Network Functions System

NFV Network Function Virtualization

NFVI Network Function Virtualization Infrastructure

NFVO Network Function Virtualization Orchestrator

NS Network Service

OAM Operations and Management

ODL OpenDaylight

OFS Open Flow Switch

OL Orchestration Layer

ONF Open Networking Forum

OP Observability Point

OPEX Operational Expenditure

OS OpenStack

OSS Operation Support System

OTT Over The Top

PAP Policy Administration Point

PDP Policy Decision Point

PEP Policy Enforcement Point

PIP Policy Information Point

PoP Point of Presence

POX Python-based software-defined networking

QoS Quality of Service

RO Resource Orchestrator

ROS Resource Orchestration Sublayer

SAP Service Access Point

SDN Software Defined Networking

SG Service Graph

SL Service Layer

SLA Service Level Agreement

SLM Service Level Measurement

SP Service Provider

SP-DevOps Service Provider DevOps

UN Universal Node

VIM Virtualized Infrastructure Manager

VM Virtual Machine

VNF Virtualized Network Function

VNF-FG Virtualized Network Function Forwarding Graph

VNFM Virtualized Network Function Manager

xDPd eXtensible OpenFlow DataPath daemon

2.2 Definitions

Big Switch with Big Software (BiS-BiS) BiS-BiS is virtualization of a Forwarding Element (FE) connected with a Compute Node (CN) node, which is capable of running Network Functions (NFs) and connecting them to the FE. See also page 12

Management and Orchestration (MANO) In the ETSI NFV framework, this is the global entity responsible for management and orchestration of NFV lifecycle.[ETS14]

Monitoring Function (MF) MF is a Network Function (NF) that implements observability capabilities in the UNIFY production environment. A MF is responsible for $i)$ control of lower-level MFs, $ii)$ collection of data, $iii)$ data processing and $iv)$ operations towards the OPs.

Network Function (NF) We use the term according to the definition of the Service Function (SF).

Network Function Forwarding Graph (NF-FG) NF-FG defines a selected mapping of Network Functions (NFs) and their forwarding overlay definition into the virtualized resources presented by the underlying virtualizer. See for more details Definition 3 on page 13.

Network Function Information Base (NF-IB) A database, which contains $i)$ resource models at networking and software abstraction of Network Functions for orchestration and $ii)$ definitions for model-based Network Function decompositions.

Network Function Virtualization (NFV) The principle of separating network functions from the hardware they run on by using virtual hardware abstraction.

Network Function Virtualization Infrastructure (NFVI) Any combination of virtualized compute, storage and network resources.

Network Service (NS) We adhere to the IETF definition: "An offering provided by an operator that is delivered using one or more service functions."

Observability Observability refers to methods measuring or estimating metrics or Key Performance or Quality Indicators in order to determine particular system states.

Observability Point (OP) OP provides a virtual context (control plane part) which operates on virtual/physical resources like counter (data plane part) and comprises of local management and access to monitoring information. See [D4.2] for further details.

Service Access Point (SAP) A service provider's physical or logical port, which represents customers' point of presence, access to internal services or exchange points to other providers. SAP definitions are included into virtualization. The description of the SAPs is part of the contract between the provider and the consumer of the virtualization. See also page 13.

Service Graph (SG) SG is an ordered interconnection of abstract Network Functions (NFs), forwarding overlays, and Key Performance Indicators (KPIs) and corresponding thresholds. An abstract NF is implementation agnostic; the forwarding overlay contains $i)$ traffic classifications and associated forwarding rules and $ii)$ Service Access Points (SAPs) or ports of NFs from other SGs. See also page 15.

Service Level Agreement (SLA) A SLA is an element of a formal, negotiated commercial contract between two Organizations, i.e., one with a provider and one with a customer role. It documents all the common understating of all aspects of the product and the roles and responsibilities of both organizations. An organization is a single legal entity, single individual or a group of people.[TM 12]

Software Defined Networking (SDN) The physical separation of the network control plane from the forwarding plane, and where a control plane controls several devices.

Service Provider DevOps (SP-DevOps) SP-DevOps is an assembly of processes and supporting tools handling technical aspects of monitoring, validating and testing programmable infrastructure.

Troubleshooting Troubleshooting refers to techniques that correlate and filter information that allows identifying a particular erroneous situation in the UNIFY production environment.

UNIFY Resource Service Resources orchestration with isolation among virtualizers and Network Function Forwarding Graph (NF-FG) requests, where resources orchestration is defined as an optimized allocation of an NF-FG request formulated against one of the virtualized views of a Resource Orchestrator (RO) to the underlying virtualized resource view. See also page 19.

Universal Node (UN) 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. A UN can be virtualized as one or many BiS-BiS (see Def. 1 on page 12) virtualization. UN is a UNIFY concept.

Verification Verification refers to procedures that compare expected versus detected system states in the UNIFY production environment.

Virtualized Network Function (VNF) An implementation of a Network Function that can be deployed on the virtualization as presented by the lower layer component. Therefore a VNF is always relative to a virtualization and the implementation may be an abstraction only. At the physical infrastructure a VNF may map to a software resource or to a hardware appliance.

VNF Development Support VNF Development Support is the SP-DevOps process that enables the creator of a virtual network function to perform development and testing tasks in a production-like environment.

2.3 Conventions

We adhere to the following conventions, some of which are also followed by ETSI NFV and ONF SDN:

abstraction is a representation of an entity in terms of selected characteristics. An abstraction hides or summarizes characteristics irrelevant to the selection criteria.

virtualization is an abstraction whose selection criterion is dedicated to a particular consumer, client or application.

reference point identifies a peer-to-peer relationship between functional blocks. The information exchanged across these reference points are modeled as an instance of a protocol-neutral information model.

interface is defined according to the functions exposed in a producer and consumer relationship. Interfaces are mapped to a reference point.

We use the following color code to distinguish different roles and responsibilities in the figures included in this deliverable:

greenish for both physical and virtualized resources;

redish for control and orchestration components at compute, storage and network abstraction; these are the most important components for the UNIFY architecture;

bluish for management- and service-logic aware components outside the Network Functions System.

black mostly for Network Functions and other, non highlighted, components.

Note, however, that components can realize multiple functionalities. As a result, the color code may vary according to the functionality illustrated. In cases where a multitude of roles takes place, we use the color that reflects the primal role of the functionality.

3 Overarching Architecture

The UNIFY architecture defines the main architectural components and reference points relevant to the UNIFY concept which allows focused work on orchestration, SP-DevOps and high performance data plane based on COTS hardware. We first present an overview of all major components and reference points organized into layers in Sec. 3.1. We continue with the description of main components in the order of their dependencies in Sec. 3.2 and with main features and benefits in Sec. 3.3. Finally, we discuss security considerations in Sec. 3.4.

3.1 Overview

The overarching view of the UNIFY architecture comprises three layers, namely, the Service Layer (SL), the Orchestration Layer (OL) and the Infrastructure Layer (IL). The architecture also includes management components, a Network Functions System (NFS) and reference points between the major components (see Fig. 1).

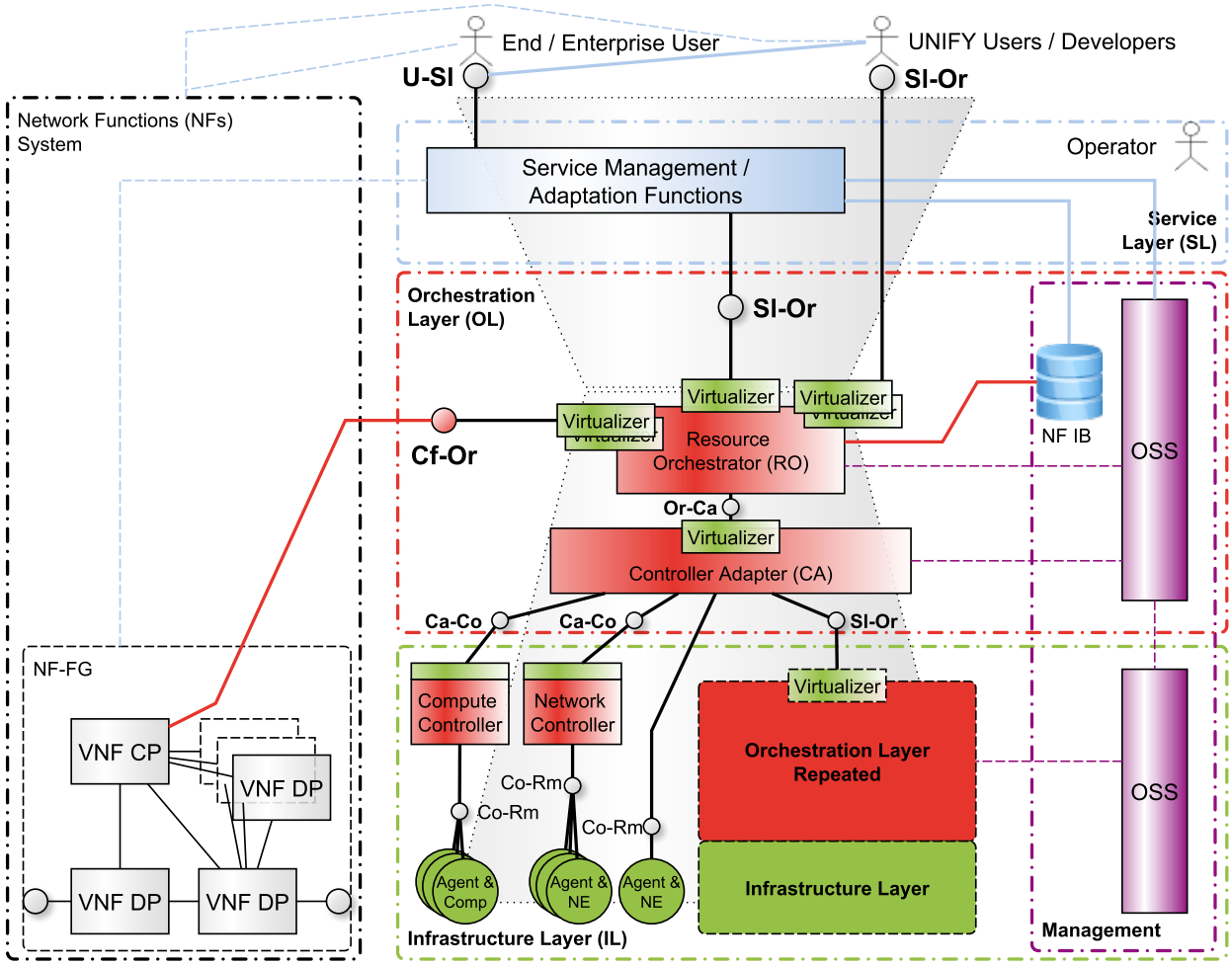

Figure 1: The UNIFY Overarching Architecture

3.1.1 Layers and concepts

We have discussed layering as one of the fundamental design principles in Sec. 5 of [D2.1]. We use the notion of layers to group functional components with similar concerns or abstractions. Our architecture comprises of three layers as

introduced in Sec. 6.2 of [D2.1]:

Service Layer (SL) comprises "traditional" and virtualization-related management and business functions concerned with service lifecycle (note that in Fig. 1 we denote all these functions with "Service Management and Adaption Functions", however their details are discussed in the context of the functional architecture in Sec. 4):

- Traditional lifecycle management functions include Element Management Systems (EMS), Operation Support Systems (OSSs) and Business Support Systems (BSSs) related to service and business management associated with Service Providers (SPs).
- Virtualization-related management functions include lifecycle management for virtualized network services, lifecycle management for Virtualized Network Functions (VNFs) and orchestration over the resources presented by the lower layer.
- Adaptation functions toward the lower layer.

SL management functions should be infrastructure-agnostic and should deal with the management of the offered services. We denote the users of the management systems by "Operators".

Users of the SL typically, but not exclusively, include:

End/Enterprise Users who consume traditional telecommunication services packaged by the SP. These services are defined with explicit or implicit Service Level Agreement (SLA)[TM 12]. The SP manages these services using OSS and BSS. If End or Enterprise Users would like to control and manage their service offerings then they can use the services offered to the other user types.

UNIFY Users comprising Retail Providers, Over The Top (OTT) Providers, Power Users, etc. who consume UNIFY Resource Service (see Def. 5) directly.

Developers who, besides consuming UNIFY Resource Services (see Def. 5) directly, can develop and test Network Functions (NFs) or package service offerings. They rely on SP-DevOps services as described in Sec. 3.3.6. Note that Developers can also be internal to the SP but, for the sake of simplicity, we only show and discuss external Developers according to Fig. 1. The different developer roles and services are detailed in [D4.1].

The Adaptation Function is detailed in Sec. 3.2.3.

Orchestration Layer (OL) comprising of two major functional components:

Resource Orchestrator (RO) comprising of virtualizers (see Sec. 3.2.1), policy enforcement (see Sec. 3.2.8) and resources orchestration between virtualizers and the underlying resources (see Sec. 3.2.7).

Controller Adapter (CA) comprising domain-wide resource abstraction functions and virtualization for different resource types, technologies, vendors or even administrations. The CA maintains the domain global view of resources and capabilities and presents its virtualization to the RO. With respect to roles and responsibilities, the CA can be regarded as a multi-technology, multi-vendor or multi-domain controller. The CA is detailed in Sec. 3.2.5.

The RO and CA are managed by a corresponding management system including, for example, by an OSS shown in the right-hand side of Fig. 1.

Infrastructure Layer (IL) comprising of resources, local resource agents and/or controllers:

Controllers comprising virtualization functions corresponding to a single domain. For example: an SDN Controller for a transport network domain; a compute controller for a Data Center (DC).

Agents and Physical Resources comprising all possible resource options (compute, storage and network) together with their corresponding local agents, e.g., OpenFlow switch agent [Ope13], Open Stack Nova compute interface [Ope14c], etc.;

Controllers, Agents and physical resources are managed by a corresponding management system such as, for example, an OSS as illustrated in Fig. 1.

Management comprising of infrastructure management functions, (virtual) NF related management information bases and management for the OL and IL. If the physical infrastructure belongs to the operator of the UNIFY OL, then management functions include physical infrastructure management as well. Alternatively, a multi-administration concept is discussed separately in Sec. 3.3.5. Part of management is a Network Function Information Base (NF-IB) used by the RO and the management components within the SL to define the (virtualized) NF-related information necessary for service agnostic resource orchestration (see Sec. 3.2.6).

Additionally, we have identified four management processes related to SP-DevOps: Observability, Troubleshooting, Verification and VNF Development Support (see [D4.1]). Key to all four processes is monitoring functionality (see Sec. 3.2.9).

Network Functions System (NFS) comprising of instantiated NFs, including data, control and management plane components and the corresponding forwarding overlays.

Note that both the CA and the RO "orchestrate" between their northbound virtualization views. However, we distinguish their roles such that the CA's primarily task is to translate and adapt to resource provider protocols, while the RO's primarily task is to orchestrate requests between different virtualization views. That is, the CA translates from one to many controller Application Programming Interfaces (APIs), while the RO translates many virtualization views to one virtualization (one-many vs. many-one).

3.1.2 Reference points

We use reference points to identify peer-to-peer relationship between "producer" and "consumer" functional blocks. The information exchanged across these reference points is modeled as an instance of a protocol neutral information model.

U-Sl between users and the SL management and adaptation functions. The corresponding interface definitions are out of the scope of the UNIFY project but our assumptions on primitives are discussed in Sec. 4.1.1. .

Sl-Or between Adaptation Functions in the SL or external consumers and the RO. The RO presents an abstract view of resources per consumer with the help of virtualizers (see Sec. 3.2.1) and accepts resource requests in the form of Network Function Forwarding Graphs (NF-FGs) (see Definition 3 in Sec. 3.2.2). The primitives associated with the Sl-Or reference point are described in Sec. 4.1.2. The Sl-Or is one of the main reference points of the UNIFY framework.

Or-Ca between the RO and the CA. The CA presents an abstract view of the topology, resources and capabilities of the underlying infrastructure (see Sec. 3.2.5) and accepts resource requests in the form of NF-FGs (see Definition 3 in Sec. 3.2.2). The primitives associated with the Or-Ca reference point are described in Sec. 4.1.3.

Ca-Co between the CA and the controllers. The Ca-Co reference point captures the various interfaces to the north of the underlying controllers, who manage different resources, technologies, vendors, domains, etc. See Sec. 4.1.5 for further discussions.

Co-Rm between a controller and a local resource agent. The corresponding interface definitions are out of the scope of the UNIFY project. We rely here on existing protocols and ongoing standardization work, like OpenFlow [Ope14a], NETCONF[Enn+11], OpenStack Compute[Ope14c], etc. See Sec. 4.1.6 for further discussions.

Cf-Or between NFs and the RO. This reference point is one of the highlights of the UNIFY framework. Through the Cf-Or reference point the OL can provide resource control functions directly to NFs deployed within the NFS. We envision, that the management of elastic services can be shared between control plane NFs and the RO. We detail the use of the Cf-Or reference point for elastic services in Sec. 3.3.3.

In Sec. 5.5 of [D2.1] we argued for the need of a new narrow waist programmatic reference point for joint compute, storage and network orchestration. Our Sl-Or reference point captures the definition of such a narrow waist with a minimalist abstraction corresponding to compute, storage and networking. The hourglass in the background of Fig. 1 illustrates such a narrow waist view.

3.2 Main components

In order to explain the UNIFY concept, we revisit here the main architectural components. Our discussion of components and information models follows their dependencies rather than the top-down structuring introduced in Sec. 3.1. Therefore, we start our discussions with the virtualizers and the NF-FG, which are used across several reference points. We continue with the SL components and the IL components, so that we can discuss in details the two main components, i.e., the CA and the RO of the UNIFY architecture. Finally, we discuss policy enforcement points and monitoring to extend the more static view of the resources.

3.2.1 Virtualizers

Virtualizers are responsible for the allocation of abstract resources and capabilities to particular consumers and for policy enforcements. This virtualized views should be vendor, technology and domain agnostic and contain the resources at compute, storage and network abstraction and capabilities. Capabilities are means to express explicit resources, for example, execution environments, list of instantiable NFs, etc. They are not meant to express resources at compute, storage and network abstraction only.

Let's assume that for the sake of simple operations, the Service Management and Adaptation Functions residing in the SL would like to rely completely on the orchestration services offered by the RO. That means, virtualization should be a mean to simplify the task of resource orchestration. For a network domain, such simple virtualization approach could be a single Big Switch abstraction. In this context let us define a compute, storage and network virtualization as follows:

Definition 1 (Big Switch with Big Software (BiS-BiS))**.** BiS-BiS is virtualization of a Forwarding Element (FE) connected with a Compute Node (CN) node, which is capable of running NFs and connecting them to the FE, as shown in Fig. 2.

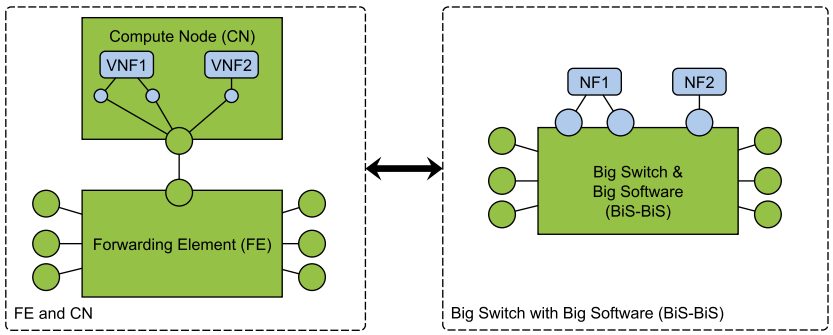

Figure 2: An illustrative example of a Big Switch with Big Software (BiS-BiS) virtualization

Let us also define Service Access Points (SAPs):

Definition 2 (Service Access Point (SAP))**.** A service provider's physical or logical port, which represents customers' point of presence, access to internal services or exchange points to other providers. SAP definitions are included into virtualization. The description of the SAPs is part of the contract between the provider and the consumer of the virtualization.

Examples of SAPs are "Port no.523 in Building X, with VLAN no.78 encapsulation", or "Public IP address 123.123.123.123 with VxLAN encapsulation id 9876".

The resource orchestration task over a single BiS-BiS virtualization is trivial: either there are enough resources to accommodate the request in the single node or the request must be rejected. The left hand side of Fig. 3 shows an RO presenting a single BiS-BiS virtualization to the Service Management and Adaptation functions.

However, a simple BiS-BiS virtualization may not be sufficient for all consumers, for example, a consumer want to have control over disjoint paths between her SAPs. One possible way for a provider to expose such capability is to create a topology of resources in which disjoint paths exist. In turn, a consumer may exercise its control regarding disjoint paths by pointing to the corresponding resources. Therefore, the topology provides means to both expose capabilities and to enforce policies. Therefore, by purpose, the RO can create arbitrary virtualization to its (other) consumers. Such virtualization can be defined by an Operator (see bootstrapping in Sec. 6.4.2 in [D2.1]), by a business contract (e.g., SLA) or may be requested dynamically through a management interfaces. The right hand side of Fig. 3 shows an example additional virtualization associated with a Retail Provider.

One should note that all the elements are uniquely identified in their provider – consumer context, so that it is possible to reference and reuse them later on. For example, in order to share an NF instance between two Service Graphs (SGs), the UUID of the NF must be used in both SGs.

3.2.2 Network Function Forwarding Graph

The NF-FG is one of the key concepts of the UNIFY programmability framework of [D3.1]. The NF-FG abstract information model is central to the UNIFY framework and it is used primarily at the Sl-Or, Cf-Or and Or-Ca reference points.

Definition 3 (Network Function Forwarding Graph)**.** NF-FG defines a selected mapping of NFs and their forwarding overlay definition into the virtualized resources presented by the underlying virtualizer. The NF-FG contains:

- the assignment of NFs to the virtualized software resources;

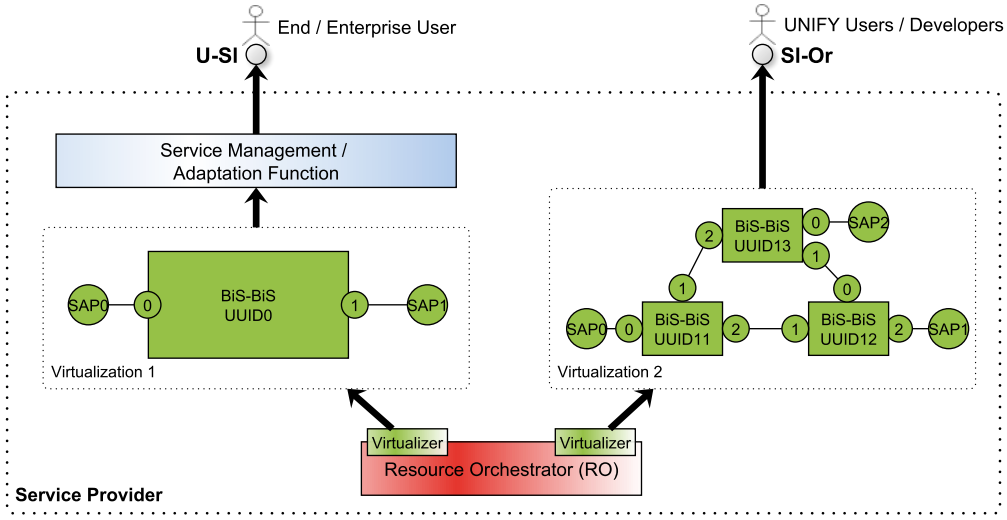

Figure 3: Different virtualization examples of an RO

- the definition of the forwarding behavior in the virtualized network resources;
- resource allocations for NFs and forwarding (optional);
- service requirements evaluable at network and software abstraction, e.g., max delay, min bandwidth, etc. (optional).

Regarding the assignment of NFs to virtualized software resources we assume that NFs can be attached to dynamically created ports (see blue NFs and ports in Fig. 4). Also, we assume that the corresponding consumer of the virtualization must fully specify the forwarding overlay between infrastructure ports (green) and dynamically created NF ports (blue). Such overlay definition can be seen in the blue call-out boxes with traffic classifications, e.g., match (&) for ({fr-a}), and corresponding actions, e.g., forward to port 1 (=>1) with tagging (Tag=B). We consider tagging as an infrastructure agnostic abstract mean to identify packets and convey information between the different virtualized resources, which will be translated according to the underlying technologies orchestrated. We also consider that such tags and ports are local to the virtualization view and can be altered by the underlying layers. The only exceptions are the SAPs, where technology specific tagging apply, which is negotiated as part of the service contract. Otherwise, Fig. 4 shows two example NF-FGs mapped to the two virtualization options shown in Fig. 3. The blue components (NFs or forwarding overlays) must be defined by the corresponding consumer of the virtualizers, i.e., orchestration must be done according to the details of the virtualization. The two NF-FGs are equal with respect to the delivered service between SAPs 0 and 1. The green ports in the figures denote ports belonging to the virtualization producer (see RO virtualizers in Fig. 3). The blue ports are logical ports defined by the consumer for deployment.

Note, that all the components within an NF-FG must have unique identifiers (references), though identifiers may belong to conceptual and not physical resources (see also discussions in Sec. 3.2.1). The detailed specification of the NF-FG and its protocol mappings are given in [D3.1].

3.2.3 Service management and adaptation functions

One of the tasks of the Service Management and Adaptation Functions is to map Key Quality Indicator (KQI) associated to the overall performance of a service/product, which is meaningful to customers, to a mix of Key Performance Indicators

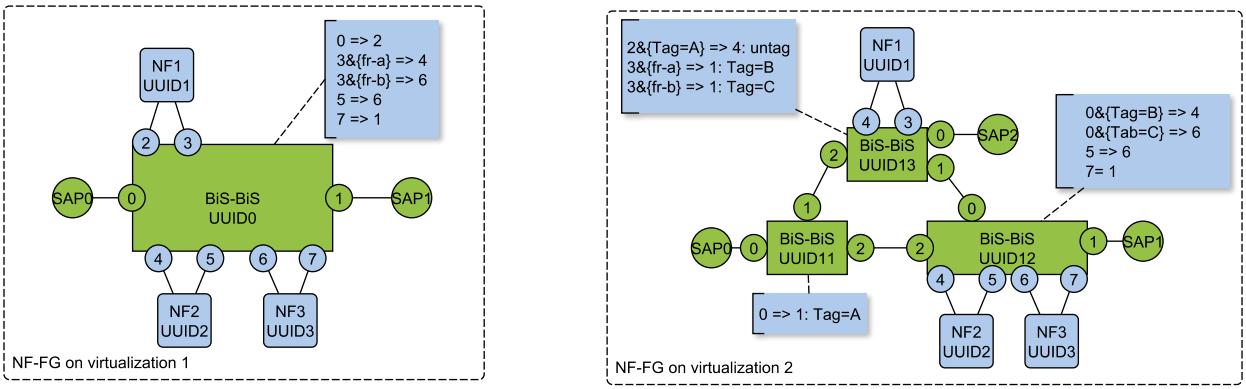

Figure 4: An illustrative example of a Network Function Forwarding Graph (NF-FG)

(KPIs). The KPIs must be associated with measurements, and measurements with measurement points. Service Level Measurement (SLM) is defined by the corresponding measurements at time t, i.e., $\mathrm{SLM}(t)$. A delivered service meets its KQI at time t if the associated $\mathrm{SLM}(t)$ measurements are all over/below their corresponding thresholds. Note that measurements may contain estimations or other processing[TM 12].

The main task of the Adaptation Functions is to translate different types of service requests into the format of the underlying layer. We introduced the definition of SG to the north of the Adaptation Function.

Definition 4 (Service Graph (SG))**.** SG is an ordered interconnection of abstract NFs, forwarding overlays, and KPIs and corresponding thresholds. An abstract NF is implementation agnostic; the forwarding overlay contains $i)$ traffic classifications and associated forwarding rules and $ii)$ SAPs (or ports of NFs from other SGs).

Fig. 5 shows a graph representation of a SG with vertices and edges. The vertices represent the ports of NFs and NFs; the directed edges together with their attributes define the forwarding overlays. The edge attributes contain forwarding rules (fr-a, fr-b in the example). The forwarding rules may contain traffic classifications and/or forwarding actions, e.g., if "voice" then assign "priority high". The direction of edges defines the output $\rightarrow$ input relationship between the ports and NFs. If no explicit edge attributes are given, then we assume that all traffic are to be forwarded between the vertices. For the sake of easier representation, we will not draw links between the ports of an NF and the NF (see for example the SG in Fig. 6).

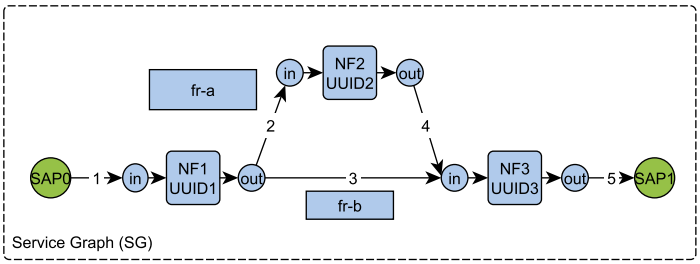

Figure 5: An illustrative example of a Service Graph (SG)

Every NF in a SG has a unique identifier (shown as $\mathrm{UUID1..3}$) used within the SL to refer to the virtual instance of the service component. This allows the SL to share NFs between SGs, for example, a Network Address Translation (NAT) NF may be shared among multiple users defined by multiple SGs.

Finally, a SG needs to be mapped to the virtualized resources presented by the underlying layer's virtualizer (see Sec. 3.2.1). If the presented resource view is a single BiS-BiS then the resource mapping is trivial. However, if the virtualization is more complex than a single BiS-BiS, then resource orchestration shall also take place in the SL. Fig. 6 shows the mapping of an SG into two virtualization views according to Fig. 3.

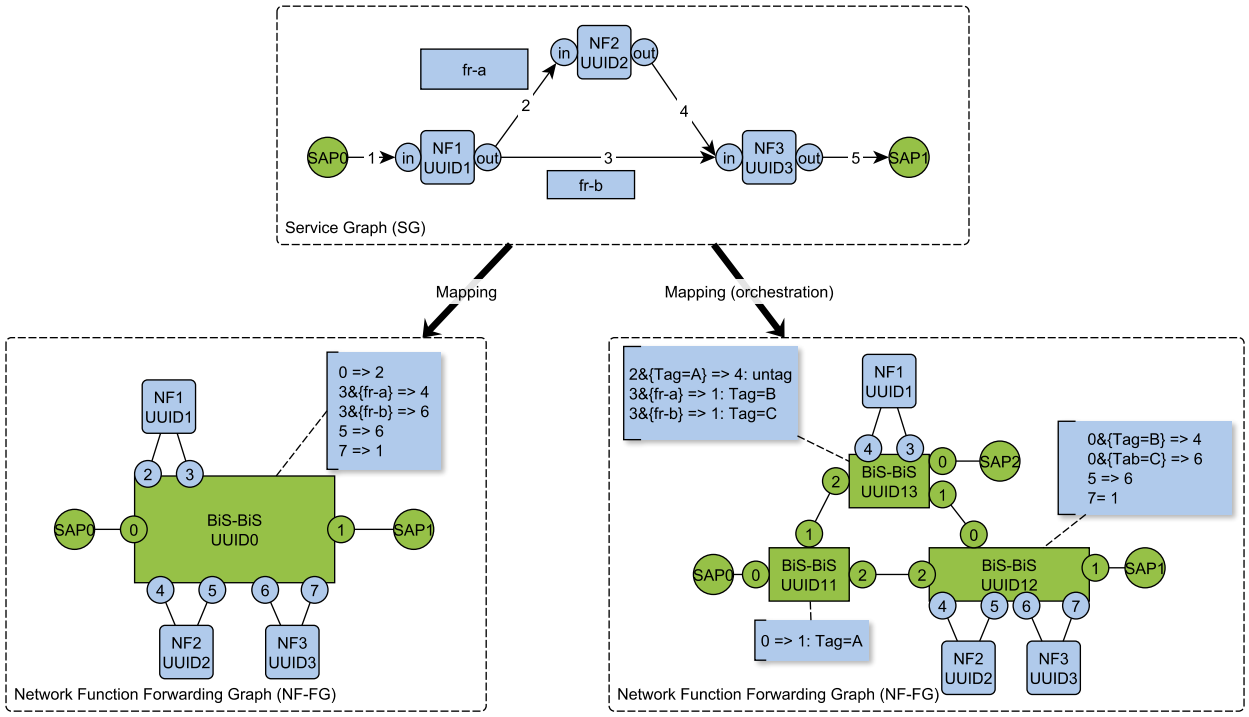

Figure 6: Exemplary mappings of a SG to NF-FGs

3.2.4 The Universal Node

To be able to design a flexible, easily programmable network, we have to prove that programmability in packet processing does not incur high performance penalty. We argue that today's Ethernet chips are already at a level of complexity where the additional overhead of programmability is relatively low. This realization may have serious impact to the unfolding of SDN. If the price of programmability is indeed low, more programmable chips may prove to be a cost-effective solution for a wide-range of task eventually paving the way towards SDN. With the introduction of the UN according to [D5.1] and [D5.2] our goals are to design and develop a COTS hardware based packet processor node that is capable of high-performance forwarding and also capable of running high-complexity NFs in its virtualized environment.

This technical approach will open up wider possibilities not only for reducing OPEX and CAPEX but also for building up new services or re-engineering existing ones with a far more flexible approach: even more this is accelerating the innovation cycles and enabling faster time to market, allowing for a higher degree of customization and, last but not least, improving telecommunication economics in terms of operations.

From resource abstraction point of view, the UN is equivalent to the BiS-BiS model at Sl-Or as introduced in Sec. 3.2.1. That is, a UN is a collection of forwarding elements and embedded software resources, albeit with high performance data plane execution environment [D5.2], under a joint Resource Orchestrator for both software and network resources. Therefore, the Northbound reference point of the UN resembles the Sl-Or reference point both virtualization and programming wise. Fig. 1's embedded OL resembles a UN as shown in Fig. 7.

The Sl-Or reference point at the North of the UN introduces a recursion into our architecture. This is further detailed in the next section.

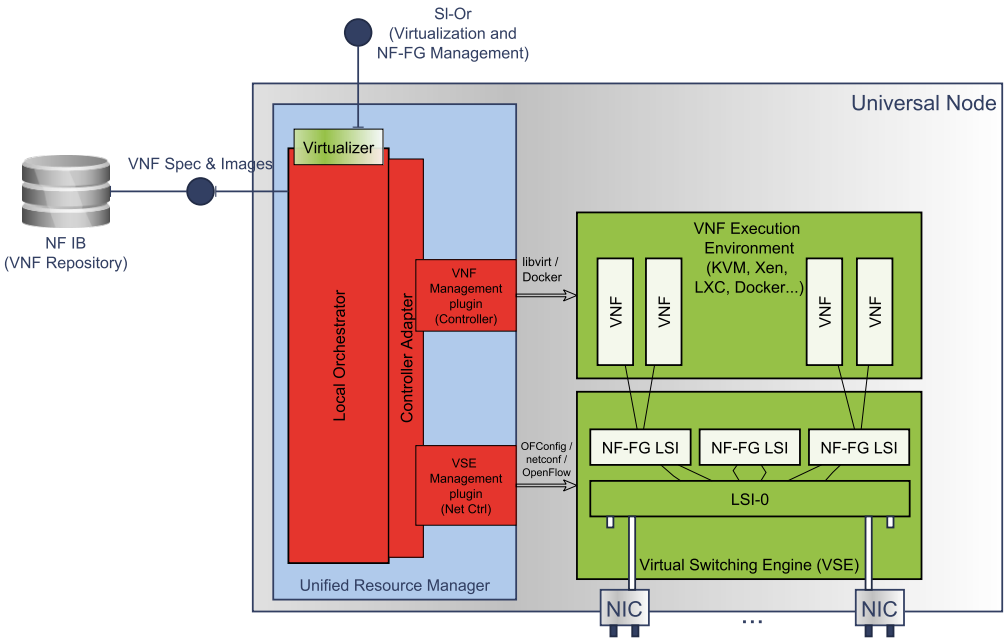

Figure 7: The Universal Node (UN) System Design

3.2.5 Controller Adapter

The CA is responsible for generating the domain-wide resource abstraction. The CA also uses a virtualizer to present the resources to the RO. However, unlike the case of Service Management and Adaptation functions in the SL, the RO wants to take full control of the resource orchestration between the its virtualizers and the underlying resources. Therefore, the role of the CA is to retain the topology, resources and capabilities learned from its controllers in its virtualization to the RO.

The bottom part of Fig. 8 shows two Open Flow Switches (OFSes) with their SDN Controller and two UNs. The SDN Controller is connected to the CA through a Ca-Co reference point, while the two UNs are connected directly through Sl-Or reference points. The CA uses resource abstraction functions and protocol adapters to interface with the underlying SDN Controller and the UNs; and uses a virtualizer to create the domain wide resource view. In Fig. 8, the domain wide view contains:

- FEs and BiS-BiS Nodes as abstractions of OFSes and UNs respectively;
- topology information between the nodes (ports and corresponding links);
- resources and capabilities (e.g., BiS-BiS vs. FE).

Notes related to Fig. 8:

- the presentation is agnostic of the underlying technologies, vendors or domains and the unique identifiers of the elements in the virtualization;
- virtualization views on the top of the RO are "independent" of the underlying virtualization views;

- the infrastructure resources could contain a DC with an appropriate controller.

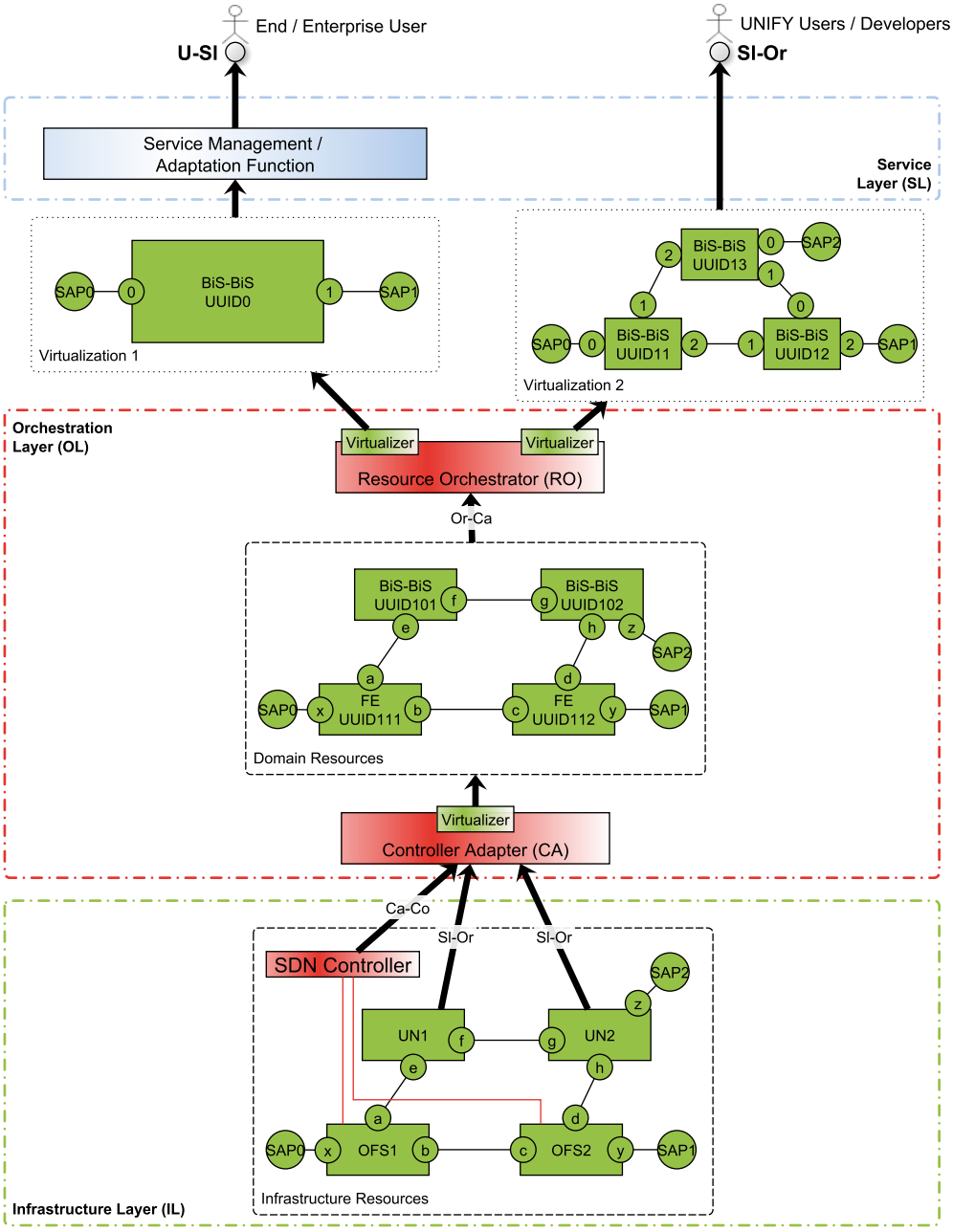

Figure 8: An illustrative example of infrastructure resources and virtualizers

3.2.6 Network Function Information Base

In order for the RO to be able to orchestrate an NF-FG request it must know the resource characteristics of included NFs. The database or catalog containing resource models for NF abstractions at networking, compute and storage resource level is called NF-IB. The NF-IB includes the following information for each NF:

- interface descriptions;
- resource models, for example, $\mathrm{CPU} = f_c(\text{traffic rate})$ or $\mathrm{Memory} = f_m(\mathrm{CPU}, \text{traffic rate})$;

- implementation information and constraints on capabilities, e.g., Virtual Machine (VM) image, revision, execution environments, etc;
- decomposition mappings (see Sec. 3.3.1 and 3.3.2).

The NF-IB resides in the management of UNIFY and logically it belongs to the SL. The NF-IB is built and managed by service logic associated with NF and Network Service (NS) template development processes of SP-DevOps (see Sec. 3.3.6). However, the RO is the primary consumer of the NF-IB (see NF-IB Fig. 1).

3.2.7 Resource Orchestrator

An RO manages virtualizers associated to its consumers. For each of its consumer the RO provides a UNIFY Resource Service as follows:

Definition 5 (UNIFY Resource Service). Resources orchestration with isolation among virtualizers and NF-FG requests, where resources orchestration is defined as an optimized allocation of an NF-FG request formulated according to one of the RO's virtualized view to the underlying virtualized resource view.

We assume that the RO's optimization targets can be configured by the corresponding OSS (purple OSS in Fig. 1) and can include any mix of objectives at software and network abstraction, like high utilization, resource costs, energy savings, etc. Let us call these as the RO's operational policies.

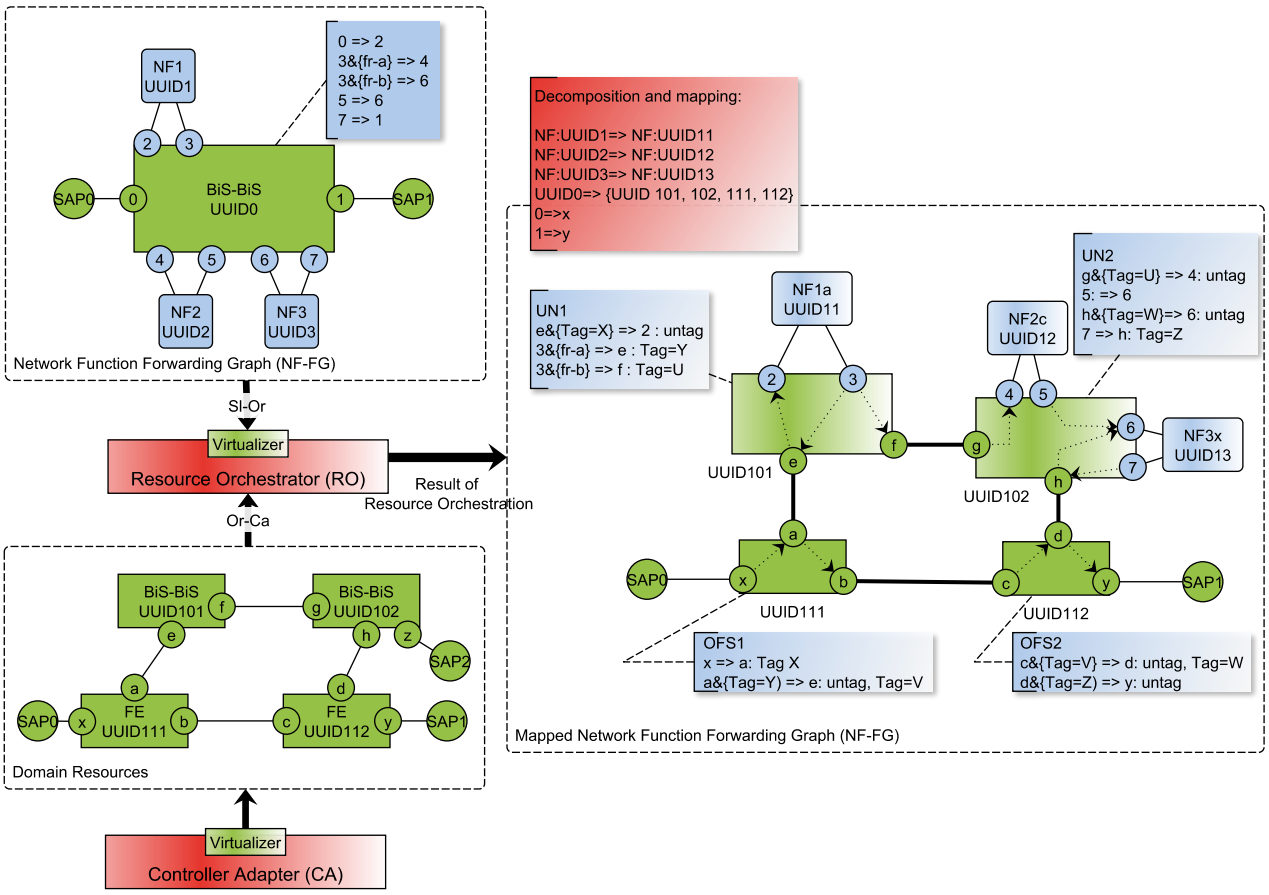

Figure 9: An illustrative example of the RO's UNIFY Resource Service

Fig. 9 shows an example resource orchestration executed by the RO. Let us assume the infrastructure and the virtualizers as shown in Fig. 8. Furthermore, let us assume that the RO receives an NF-FG corresponding to the left hand side mapping of a SG as shown in Fig. 6. Now the RO needs to orchestrate the NF-FG received at Sl-Or considering the virtualization presented to this consumer to the virtualized resource view received from the CA. The output of the resource orchestration is an NF-FG, where the NFs and the forwarding overlay definition is mapped to the resources presented by the CA. Note, that an abstract NF of the SL might translate to a different NF abstraction in the RO with the help of the NF-IB. Such translation is shown by the different NF types and UUIDs, e.g., NF3@UUID3 $\rightarrow$ NF3x@UUID13, where NF3 and NF3x are deployable NF types and UUIDs are unique resource identifiers.

Additionally, Fig. 10 shows the output of the resource orchestration forwarded to the CA. The CA, in turn, splits the NF-FG into sub-graphs corresponding to the different resource providers from the IL. If the underlying resource provider is a UN then the sub-NF-FG is forwarded directly to the UN's RO through a Sl-Or reference point. If the underlying resource provider is a controller/agent, then the corresponding sub-NF-FG must be translated to the Northbound API of the controller/agent.

3.2.8 Policy enforcement

Nowadays, an adequate use of resources is always welcome not only from an economic point of view but also from an environment aware one. Unfortunately, an uncontrolled use of resources can be not only the result of an erroneous management operation but be part of an attack, by any internal or external user of the system, as well. Any service-critical architecture should provide protection against an accidental or intentional resource starvation situation. In this sense checking the correctness of the petition, the identity and rights of the requester and the available resources before granting the use of it is not only beneficial from a economic point of view but is also a cornerstone for the stability of the service provided. With this in mind a resource policy based approach is the most adequate and flexible solution tool to cope with this problem.

The resource policy approach in UNIFY is focused on the Sl-Or and Cf-Or reference points, as these are the two sources for resource requests. As a base for defining the functionalities to be performed by each component of the UNIFY framework, we have followed the elements defined in [Vol+00] and extended in [XACML] detailed below and described in Fig. 11:

Policy Administration Point (PAP) is the repository where the policies are defined and provides them to the PDP.

Policy Enforcement Point (PEP) receives the requests, evaluates them with the help of the other actors and permits or denies the access to the resource.

Policy Decision Point (PDP) is the main decision point for the access requests. It collects all the necessary information from other actors and concludes a decision.

Policy Information Point (PIP) is the point where the necessary attributes for the policy evaluation are retrieved from several external or internal actors. The attributes can be retrieved from the resource to be accessed, environment (e.g., time), subjects, and so forth.

We consider two aspects of resource policies with regards to the mapping of policy functionalities to the UNIFY framework:

Identities Policies will define the resource usage allowed for a certain identity (could be a UNIFY User, a Control NF, etc.), so each request must be linked to the correspondent identity to be able to evaluate the policy.

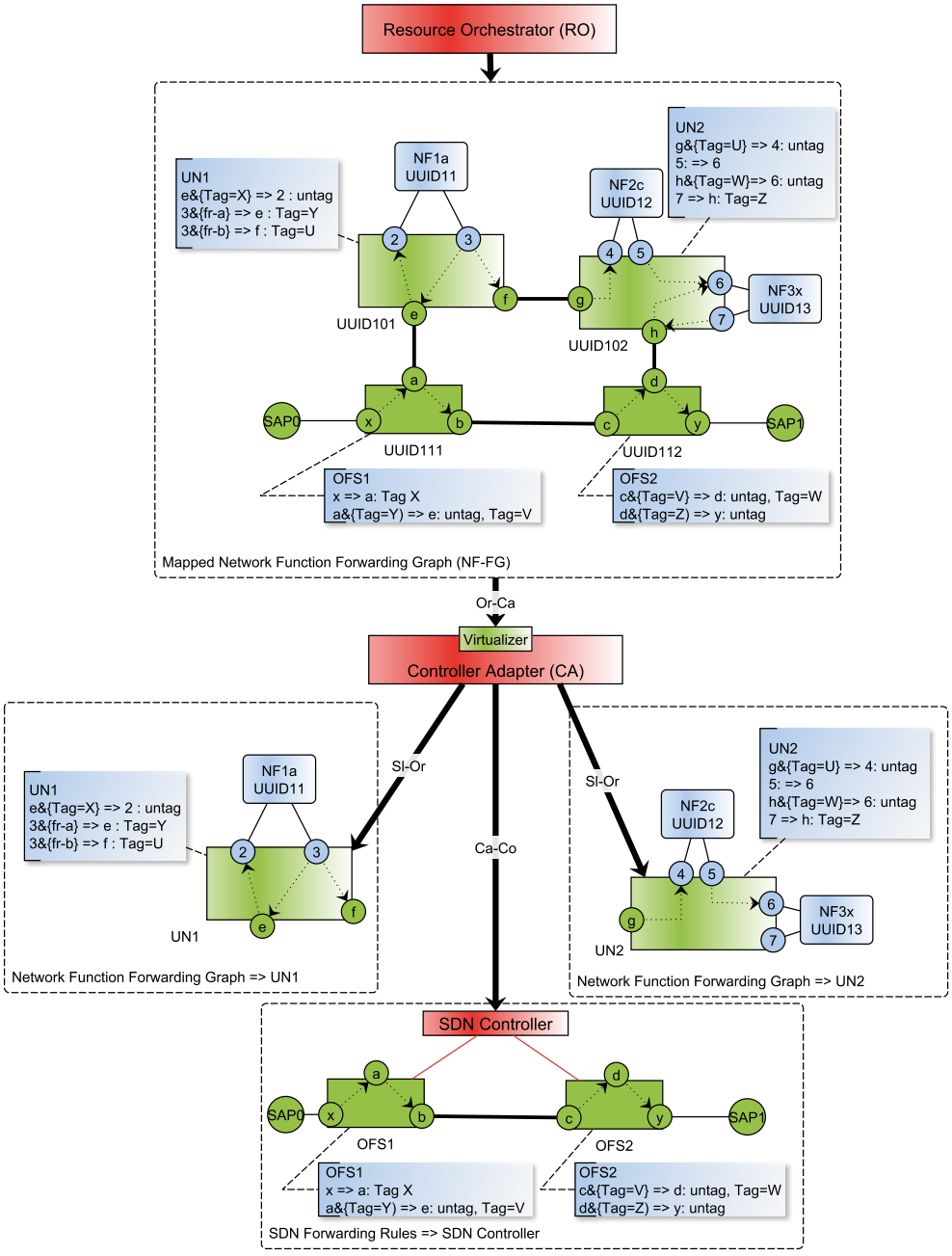

Figure 10: An illustrative example of CA's split of NF-FG into sub-domains and protocol translations to the underlying controllers/agents

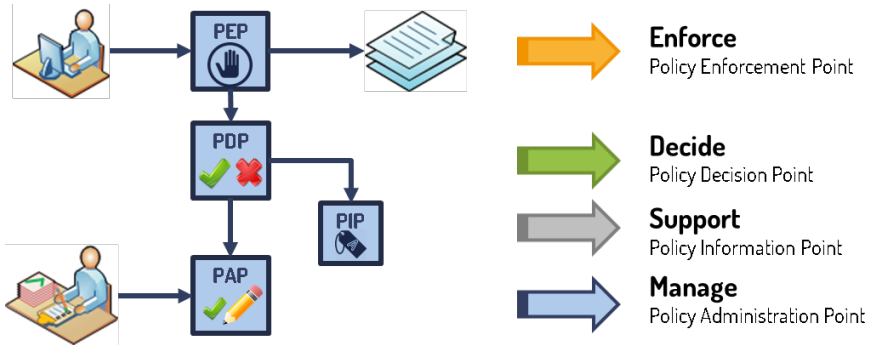

Figure 11: Main elements in XACML Policy management model

Resource domain Policies will define the resource usage allowed in a certain domain, so each evaluation of the policy must consider the situation across the whole domain.

The functionality of the PAP maps naturally to the management components and would be part of the OSS, which holds management information; has a complete, domain-wise view; and eases the integration with accounting and back-office processes.

The enforcement of the policy must be done whenever a new request for resources is received, so the functionality of the PEP must be placed in the RO, and more precisely in the virtualizers, as they interface with the consumers. This placement also allows to cover both Sl-Or and Cf-Or with a common approach. Due to the functionalities already present in virtualizers, e.g., protocol agent, PDPs could also be executed there.

Following the two aspects of resource policies, we envision two PIPs: one related to identity information located in the OSS for similar reasons as for the PAP placement; and one related to resource usage located in the CA, as responsible of the domain-wide resource abstraction.

Figure 12 shows the mapping of the resource policy functionalities to the components of the UNIFY architecture.

Considering the support for elastic services and the recursive architecture envisioned in UNIFY (see Sec. 3.3.3 and 3.3.4 respectively), as well as the design of the UN in Sec. 3.2.4, it becomes apparent that a mechanism to allow for autonomous operation of the RO at the different levels is required, to avoid hindering the operational efficiency of such approaches while at the same assuring that the global policy is still enforced.

Such a mechanism could be based on the delegation of resource usage quotas whenever NF-FG is deployed through the Sl-Or interface. This way the lower level PDP could make autonomous decisions on the resource requests as long as they meet the delegated resource quota, and the upper level PDP would still ensure that the global resource limit in the policy is met, controlling the aggregation of the delegated quotas. When the delegated quota is exceeded, one approach would be for the lower level PDP to request a quota extension to the upper level PDP. The upper level PDP could grant the quota extension, if the aggregation of the delegated quotas doesn't exceed the global policy limit or triggering a quota redistribution among the different lower level PDPs with a delegated policy quota if required, or deny it, if the global policy limit is fully covered and quotas could not be redistributed.

3.2.9 Monitoring

Beside orchestration and programming of services and resources, monitoring is one of the most essential building blocks of the UNIFY framework in terms of maintaining performance of a service and rapid reaction to infrastructure failures. In this regard, monitors collect a wide range of information that indicates performance and availability of system components, e.g., VNFs.

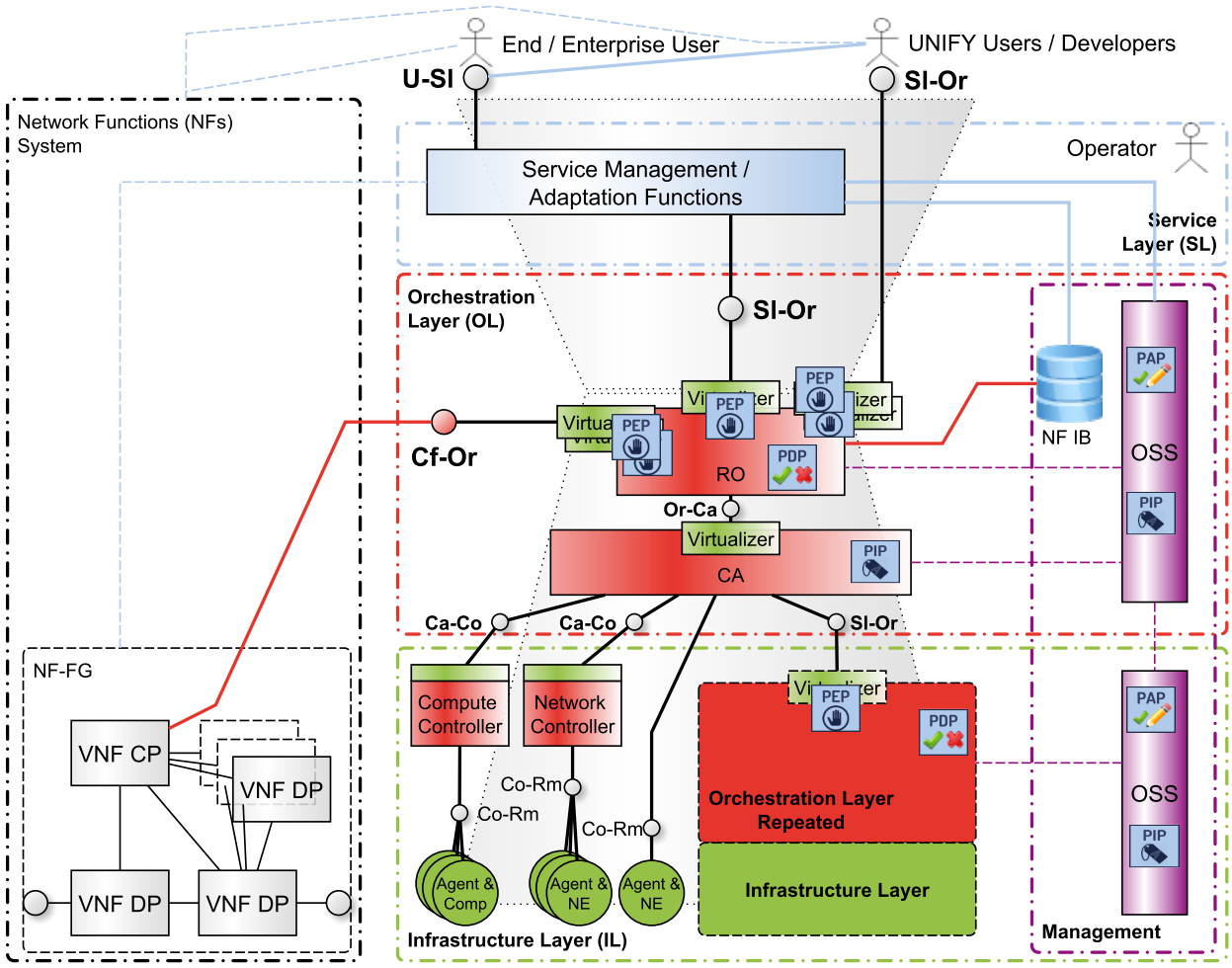

Figure 12: Resource policy functionalities in the UNIFY architecture

Understanding Monitoring Functions (MFs) on a big scale, they consists of the probing of performance properties, e.g., latency, utilization of resources, or processing and analysis of monitoring data and measurements, e.g., throughput, bandwidth utilization, energy consumption, or alarm of resource shortages. In SP-DevOps, MF addressing research challenges identified in [D4.1] and information flows are provided as part of the Observability process.

There are three different monitoring scopes:

- Service specific monitoring with monitoring elements inside the service, which not rely explicitly on infrastructure related monitoring results;
- Infrastructure monitoring for operating the three different resources based on available monitoring capabilities;
- Correlation of service and infrastructure monitoring results for improved operation by the means of a monitoring orchestrator.

The first one can be performed by inserting appropriate monitoring functions into service definitions. The second one can be implemented by sending information into an OSS system and by performing appropriate infrastructure management actions. The third scope is one of the holy grails of operating networks and needs concepts on correlating service requirements for infrastructure monitoring with appropriate translation and abstraction in-between. In the remainder of this section, we focus on describing the third option as it is a crucial part of a unified production environment.

MFs are implemented in all the layers (see OSS in Fig. 1). In line with [D4.1], it is assumed that a MF is comprised of as a standalone or embedded part of an NF-FG detailing monitoring NFs and Observability Points (OPs):

- A monitoring NF's is responsible for i) control of lower-level monitoring NFs, ii) collection of data, iii) data processing and iv) operations towards the OPs.
- An OP provides a virtual context (control plane part) which operates on virtual/physical resources like counter (data plane part) and comprises of local management and access to monitoring information. An OP can be comprised of various local Observability managers. This is similar to the virtualization of compute or network resources in the ETSI Industry Specification Group (ISG) NFV definition.

Each time a service layer is involved, MFs and quality indicators needs to be translated into NF-FG elements. Thus, the resource service offered from the orchestration layer to the service layer has to provide and indicate the capabilities for monitoring as well. This can be done in principle in two different ways: as a standalone or embedded NF-FG. The standalone option involves building/defining an NF-FG which consists of monitoring specifications only. The receiving element of the monitoring specification would be either the resource orchestrator or a dedicated monitoring orchestration element. The embedded NF-FG option consists of both service and monitoring related aspects which are further decomposed in the NF-FG receiving element. In both cases, the receiving element of the standalone NF-FG or the monitoring relevant part of the embedded NF-FG must be forwarded to an element understanding the specifications of MF. There the NF-FG must be translated and mapped to NFs (operating on information from OP) and existing, exposed OP in the infrastructure by the orchestration layer. Annotations of the NF-FG in terms of monitoring functionality, as described in [D3.1], will provide input to the translation and mapping processes.

The decomposition process is illustrated by an example of a packet counter (e.g., a base service for monitoring data throughput) implemented as a standalone NF-FG and represented in Fig. 13. The packet counter reads packet counts corresponding to a BiS-BiS virtualization (offered to the Service Management and Adaptation Functions in a SP) as shown in right side of Fig. 13, the service layer translates and formulates a monitoring NF-FG request containing i) OPs

(OBS1 and OBS2) attached to the SAPs of the BiS-BiS and ii) a aggregating monitoring NF (Mon), which provides a monitoring service aggregating the throughput of ports corresponding to the BiS-BiS virtualization. The left hand side of Fig. 13 shows a SG definition for flow counting monitoring and the right hand side shows the corresponding NF-FG definition as mapped to the single BiS-BiS virtualization example given in Fig. 2. Note, for simplicity, the edges of the SG in the left side of Fig. 13 contain bi-directional links, though the forwarding definition in the NF-FG is given for both directions separately. It is assumed that port d@Mon is using MAC based L2 forwarding to ports c@OBS1 and c@OBS2. After its successful deployment the flow counters can be read by the consumer through VLAN=A@SAP1. A separate VLAN is only used to separate user plane and management plane traffic for the customer. Also, since the SAP is technology specific, we assumed the user requested this VLAN based separation according to the definition of the SG. The consequence of the monitoring is, However, that any further NF-FG requests must be mapped between ports 3@OBS1 and 7@OBS2 in order to be able to count the corresponding traffic, i.e., these ports will become the new endpoints (or SAPs) for the consumer to use for service deployment as shown in the right side of Fig. 13. Therefore, OBS1 and OBS2 become an integral part of the NF-FG throughout the orchestration process.

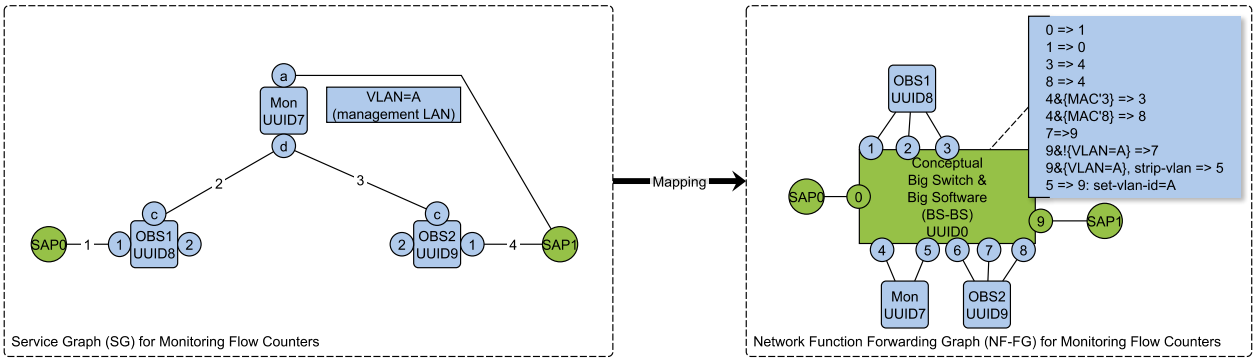

Figure 13: A flow counter monitoring example

In the orchestration layer, the resource orchestrator resolves where and how to instantiate the requested resources. Here a number of options exist with regard to MFs:

1. RO interprets MF specifications, selects the appropriate resources and attaches the monitoring information and associated behavior of NFs to the NF-FG.

2. RO has a monitoring orchestration part and transfers the monitoring part of the embedded NF-FG or the standalone NF-FG into this block. The monitoring orchestrator decomposes this monitoring related part of the NF-FG, setups or instantiate the monitoring NFs with the help of the RO or the CA and with the help of the instantiated monitoring NFs performs the monitoring for the NF-FG.

3. RO uses the OSS to perform the monitoring by transferring the standalone NF-FG or the monitoring relevant parts of the embedded NF-FG to the OSS. The OSS has control over all the monitoring NFs and observability points and performs the orchestration of monitoring information as requested by the RO as such and reports / performs actions like, e.g., reconfiguration in failure modes or requests for resource re-provisioning to the resource orchestrator only.

The second option is seen from the RO as a receiver of an NF-FG. Given the support of virtualization context, the demand for resources to instantiate NFs, the interaction between RO and monitoring orchestrator needs similar primitives as the Cf-Or reference point.

Currently all three options are under investigation, and will be further analyzed in [D4.2]. Part of the work includes further understanding of the virtualization context and synchronization between the RO and monitoring orchestration, as well as virtualization and deployment of MFs and associated OPs.

Finally, the infrastructure layer provides the OP and their information towards the orchestration layer.

3.3 Main features

3.3.1 Model-based service decomposition

We introduced model-based service decomposition in order to be able to re-use and build services out of elementary (or atomic) blocks. Our decomposition models are service specific and are determined in design time of the NFs or SGs. However, the corresponding decomposition rules driving the resource orchestration process are interpreted service agnostic by the means of NF-FGs or more specifically:

The model-based service decomposition allows for a step-wise translation of high-level (compound) NFs into more elementary or atomic NFs, which can eventually be mapped onto the infrastructure. The decomposition model is also stored in the NF-IB as a set of decomposition rules associated with NFs. The decomposition rules define logically equivalent realizations of an NF. The decomposition is given

Definition 6 (Model-based service decomposition)**.** ~ is defined as a mapping of an NF into a set of NF-FGs, more specifically, $\mathrm{NF}_i \rightarrow \{\mathrm{NFFG}_1^i, \mathrm{NFFG}_2^i, \ldots\}$. During the decomposition of an NF, the external interfaces remain unchanged.

The details of decomposition processes and methods are described in [D3.1].

3.3.2 Monolithic vs. decomposed network functions: control and data plane split design

NFV exploits only half of the opportunities enabled by compute and network virtualization. The SDN abstraction of the forwarding behavior enables the separation of traditional monolithic control and data plane network function designs.

For example, an Intrusion Detection System (IDS), whose role is to identify and block malicious traffic, could be implemented in various ways. Let's assume that the IDS service is managed by the operator, i.e., the SP manages and operates the IDS upon the user's subscription to the service. The user's and the service's view of the IDS is shown in the left hand side of Fig. 14. The ports of the NFs are shown with tagged circles when we reference to the same ports throughout the decomposition process. Small circles without tag denote shared ports and line ends shows unshared ports. The IDS's port E is connected to an Element Management (EM), which connects to an OSS for Operations and Management (OAM). Port V is optional and may be connected to a VNF Manager (VNFM) according to the ETSI Management and Orchestration (MANO) framework. We will use this component when we revisit this example in Sec. 6.3. Ports 1 and 2 denote the user plane ports. The IDS functional block can be realized as a hardware based monolithic component (a); a monolithic IDS VM (b); or as a data and control plane split design according to options (c) and (d).

In option (c) the IDS control logic is separated into an IDS Control VM (IDSC), a Firewall (FW) (FW) component for blocking of the malicious traffic and a traffic analyzer. The FW may be mapped to a FE ($\mathrm{FE1}$) and the traffic analysis is realized by the means of a generic Deep Packet Inspection (DPI) VM ($\mathrm{VM\ DPI}$) component. The IDSC must configure the generic $\mathrm{VM\ DPI}$ with patterns to identify malicious traffic. Yet another logical FE ($\mathrm{FE2}$) is used to mirror user traffic to the DPI.

In option (d), without loss of generality, we consider that the DPI processing can be parallelized by forking additional $\mathrm{VM\ DPI}$ instances. The role of the control application has also to be changed. The Elastic IDSC ($\mathrm{E\text{-}IDSC}$) must monitor

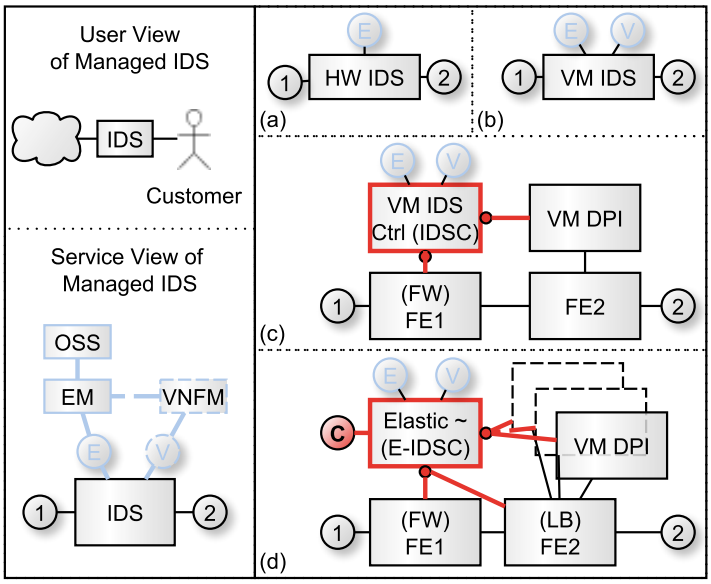

Figure 14: Model-based Service Decomposition: An IDS example

the load of the VM DPI components; add or remove VM DPI instances based on the demands and re-configure traffic steering. For example, in option (d), the FE2 will not only mirror the user traffic to a single VM DPI instance but must act as a Load Balancer (LB) among the instances of VM DPIs. Therefore, the FE2 must be dynamically configured by the E-IDSC. In order for the E-IDSC to self manage its corresponding NF-FG, a similar interface as offered to RO consumers through the Sl-Or reference point may be offered to this control application. Let port C denote the interface of E-IDSC to be connected to the Cf-Or reference point of the RO. Port C is a new external interface, whose presence depends on the decomposition option and it must connect to the RO itself, who executed the decomposition. Therefore, we assume that i) these ports are tagged distinguishably; and that ii) the RO upon discovering such tagged port during the decomposition process, it creates a corresponding virtualizer and connects the port to the virtualizer (see discussions in Sec. 3.3.3).

We would like to emphasize, that the model-based service decomposition is a recursive process, i.e., there may exist further decomposition models for an NF within a decomposition given by an NF-FG.

3.3.3 Elastic services and the Cf-Or reference point

The possibility to dynamically scale network services at run-time in an automated fashion is one of the main advantages offered by the NFV approach, providing both better resource utilization and better service at a lower cost. Elastic NFV could be similar to what elastic cloud services provide for compute, with "pay as you use" cost models for the customers.

There are multiple reasons for initiating a scaling procedure: a user could request higher capacity ahead of time in order to deal with a known increase of demand in the future; the operations and management system could initiate scaling to maintain service level requirements based on monitoring KPI and resource use; or the deployed service components themselves could manage their resource needs similarly to automatic multi-threading scaling of some software running on multi-core CPUs.

Scaling of VNFs can be done in many ways. Which one is appropriate in a particular case depends heavily on the precise function provided by the VNFs, for example, the type of traffic it operates on and at which layer in the networking

stack; requirements on state synchronization; requirements on control traffic during the scaling event; dependencies on other VNFs (service logic); ability to parallel processing or multi-threading; etc.

Regarding elasticity, in option (a) one can consider only sharing of the hardware (HW) resources, in options (b) and (c) one can scale up/down the service dynamically by requesting or releasing resources associated to the individual components, e.g., requesting more CPU for the VM DPI instance. However, scaling out/in requires adding to or removing from components and also re-configuring of the service chain.

In option (d), we assume that the fate of controlling service elasticity is shared between the service developer and the orchestration framework. That is, the E-IDSC knows the service specific logic determining how to scale up/down or in/out the service components it is responsible for, but the resource allocation is performed by the resource provider. Fortunately, a similar resource orchestration service is already available at the Sl-Or reference point for established virtualizers.

Service wise, the Cf-Or elastic control reference point offered to control applications in the NFS is equivalent to the Sl-Or reference point. However, the major differences between Cf-Or and Sl-Or are that:

- the Cf-Or virtualization must be created autonomously upon NF instantiations when detecting a named dedicated interface (see port "C" in Fig. 14);
- the connection to the corresponding Cf-Or reference point must be inserted into the NF-FG autonomously; and
- the virtualization must be strictly scoped to the subset of the NF-FG belonging to the control application.

The virtualization view is derived based on the NF mapping and decomposition models in order to prevent interference between control applications within the same genuine NF-FG.

The above IDS-based elastic service example is further elaborated and is compared to an ETSI MANO deployment scenario in Sec. 6.3.

3.3.4 Recursive orchestration

On one hand we argue that compute virtualization in general implicitly involves network virtualization too, for example, an OpenStack DC includes Neutron networking, a CN has internal network with logical switches, etc. On the other hand, if one starts with an SDN network and adds compute resources to Network Elements (NEs), then compute resources must be assigned to some virtualized network resources if offered to clients. That is, we observe that compute virtualization is implicitly associated with network virtualization. Furthermore, virtualization leads to recursions with clients (redefining and) reselling resources and services (see for example [HZH14]).

We argue that given the multi-level virtualization of compute, storage and network domains, automation of the corresponding resource provisioning needs a recursive programmatic interface. Existing separated compute and network programming interfaces cannot provide such recursions and cannot satisfy key requirement for multi-vendor, multi-technology and multi-provider interoperability environments. Therefore, our UNIFY architecture exploits a recursive programmatic interface for joint compute, storage and network provisioning.

We have revealed in Fig. 1 and in the discussions of the UN in Sec. 3.2.4 that the Sl-Or reference point appears both at the North of the RO and at the South of the CA. There are no architectural limitations on how many UNIFY domains or UNs can be put together in a multi-level hierarchy. One natural way to end-up with a multi-level hierarchical system is by joining together several horizontal businesses into a vertical UNIFY hierarchy.

Fig. 15 shows a multi-level hierarchy example with physical resources and UNIFY domains. Notice that infrastructure resources can already contain virtualizers hence allow isolation and independent control of their resources.

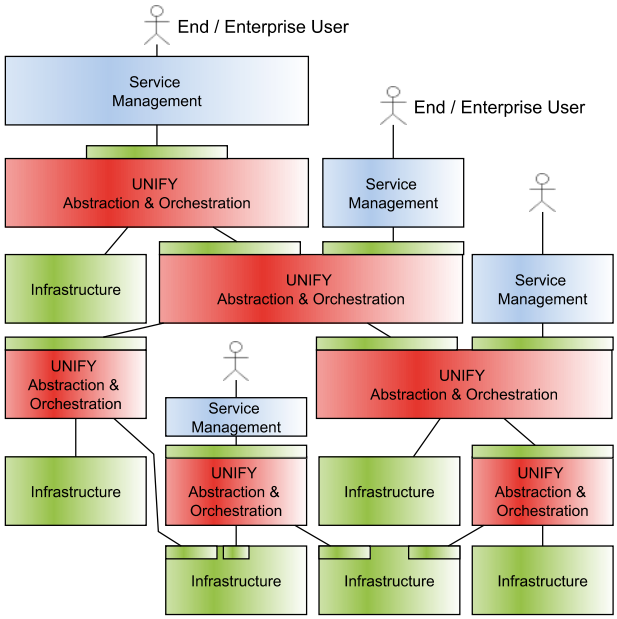

Figure 15: An illustrative example of recursive resource orchestration

3.3.5 Multiple administrations

The possibility to clearly separate roles and responsibilities in an architecture allows not only technical separations (see Sec. 3.3.4) but the possibility build a multi-level hierarchical system with more than one business actor. Such vertical business separation (and value chains) could come along with, for example, infrastructure ownership, wholesale and retail operations, OTT providers, etc.

Let us investigate what happens when infrastructure resources are split according to multiple vendors, technologies, administrations or businesses. Assume that the IL of Fig. 1 is split into different administrations[1] as shown in Fig. 16: IL@SP1 and IL@SP2 belong to two different SPs, and that IL@SP0 belongs to SP0, whose OL and SL is shown in the upper part of the figure. The NF-FG within the NFS shows that the different orchestrators can see different deployment realizations corresponding to their position in the multi-level hierarchy. The OL@SP0 sees a NF-FG comprising of VNF1 CP, VNF1 DP and VNF2, however, the later deployment of VNF2 is decomposed into a NF-FG comprising of VNF2a DP, VNF2b CP and various number of VNF2c DP components. Note, however, that the services and external interfaces of the NF-FG at SP1 must be equivalent to an abstract VNF2 at SP0.

One consequence of the multiple administration is that management systems became separated (see Fig. 16. This affects how the virtualization services from the underlying resource providers are requested. We assume that such configuration is part of the infrastructure bootstrapping processes (see [D2.1]). How such virtualization service can be requested on-demand is left for future work dealing with business-to-business aspects.

Once, the underlying virtualization services are configured, the operational orchestration interfaces corresponding to Ca-Co and Sl-Or are the same as discussed.

3.3.6 Developer support (DevOps)

In [Mei+14; D2.1] we presented the set of DevOps principles and processes that form the base for the SP-DevOps concept developed in UNIFY and outlined in [D4.1] and further refined in [D4.2].

[1] May belong still to the same ownership.

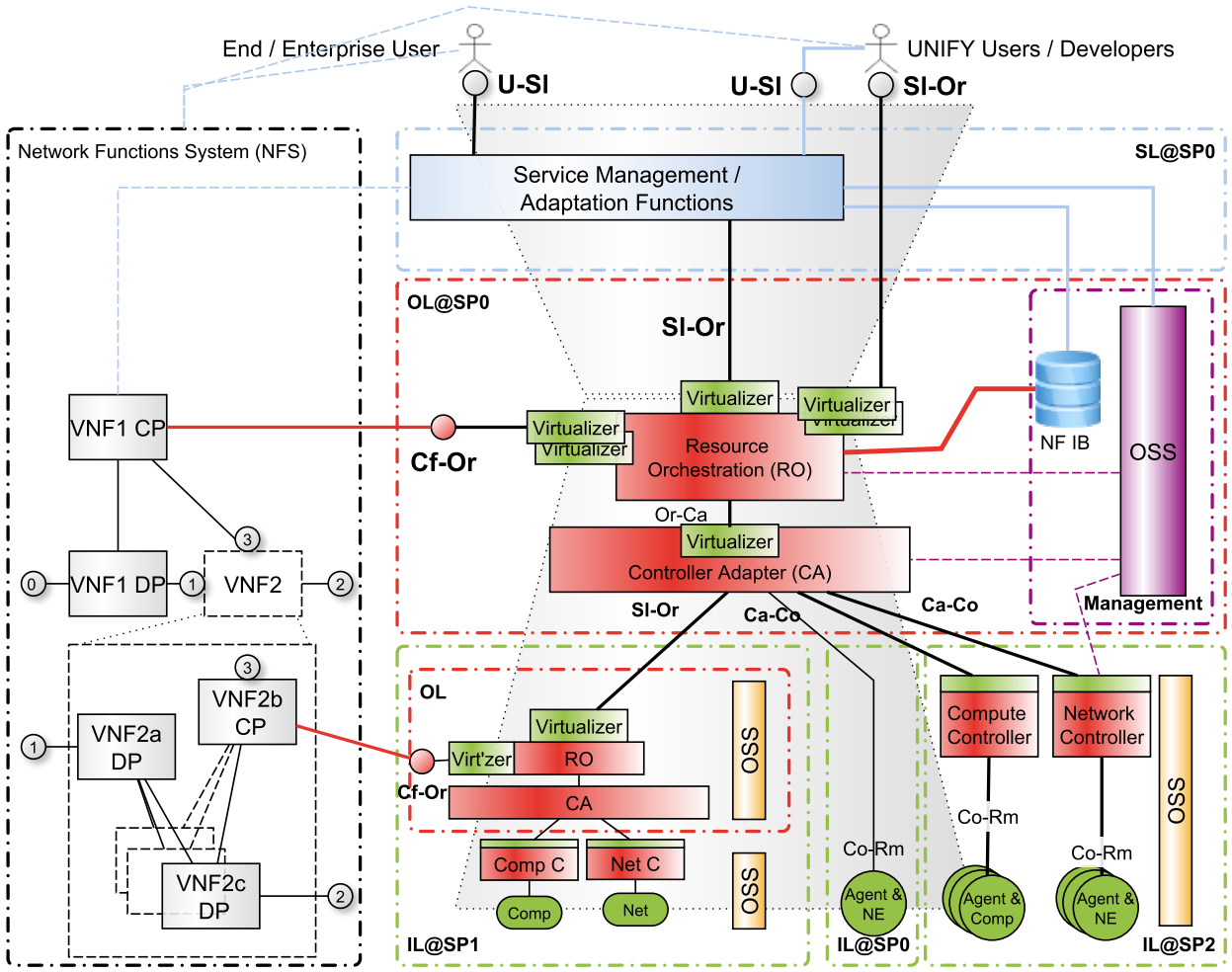

Figure 16: An illustrative example of the UNIFY architecture with multiple administrations

For resources exposed through the UNIFY Architecture, we consider that the processing of the SG within the Service Layer is equivalent to a DevOps build step, the transformation of the NF-FG within the Orchestration Layer(s) is equivalent to an DevOps orchestration step and finally the setting up of resource configuration parameters within the Infrastructure Layer(s) is equivalent to a DevOps deploy process.

The build, orchestrate and deploy processes are expected to be based on deterministic algorithms that once presented the same input and the same state of the resource allocation and consumption maps, will provide the same response. Therefore, the repeatability in the DevOps principle is achieved. The SP-DevOps Verification process defined in WP4 is expected to check intermediary transformations of the NF-FG for policy constraints, thus creating an environment that inspires further confidence regarding the correctness of operation and contributes towards achieving the reliability requirement from the DevOps principle.

Virtualizer components of the UNIFY Architecture play a key role in ensuring that the build, orchestrate and deploy processes are identical in the development, test and production environments. It enables Service Developers to first try their service on an isolated resource slice of the infrastructure, possibly as a trial with a small number of friendly users. It also enables them to simply redeploy the SG in the production environment, using the same reference points and API calls as in the case of the isolated test environment. As the SGs and NF-FGs are to be described in formal languages, the cloning of existing resource sets supporting a service in a production environment and re-creating their configuration

in an isolated slice in order to perform in-depth investigations without affecting the production system is an operation that requires very little effort on the Service Developer.

The UNIFY Architecture supports monitoring throughout all the reference points between the layers by specifying a set of functions that handle monitoring information [D3.1; D4.2], see section 3.2.9.

In addition, the UNIFY Architecture offers a Verify API call between several layers of the architecture [D4.2] that could be used to trigger tools that validate certain properties on the data plane, such as reachability, the absence of forwarding black holes, etc. UNIFY WP4 is also developing a number of tools that could be employed for checking such properties [D4.2]. In addition, SP-DevOps defined the Troubleshooting process as a capability to automatically generate and execute checklists or workflows.

While a VNF Developer may have the in-depth knowledge to control in detail or understand the messages generated in a high-verbosity operating mode of a particular tool, a Service Developer may order the execution of the tool through the SP-DevOpss Troubleshooting process, determine whether a problem is present and hand over partial results as well as all the information regarding resource allocation to the VNF Developer. In turn, the VNF Developer encounters a familiar vocabulary and interface to interact with the system, which has the potential to speed-up the process of reproducing the problem either in the production or in a test environment.

3.4 Security considerations

As many other research projects UNIFY relies on several technologies, so establishing a baseline regarding the security aspects is critical to define where efforts need to be done to assure a coherent result. To do so, we have defined three categories, each requiring a different approach.

First, we have those technologies that can be considered legacy, in the sense that they are well known and established software or hardware developments that are used unmodified. In many cases, the technologies themselves have implemented their own security analysis and put in place the tools and methods to get the required security functionalities, and no more than that is required. For example, this could be what happens with some operating systems: Independently of the actual configuration, there is SELinux (Security-Enhanced Linux) or Bastille to offer almost any kind of functionality needed. The same applies, for example, to secure booting (by using UEFI Secure Boot).

Second, there are some other technologies that are themselves actually studying the implications and are currently investigating and trying solutions. An example of this is the management of trust and compliance in OpenStack, for which different views and solutions exists. Another example is how to load and distribute public and/or symmetric keys in the VM images. In this case, the need of these features should be flagged at the architecture level, stating that deployment in a production environment should implement these required features.

Finally, there are new architectures, developments, concepts and workflows that are produced as a result of the project. In the case of requirements that directly arise from the investigation done at the project level a clear response needs to be provided. These requirements can either produce modifications in existing features of some solution used, or be addressing some previously unseen problems. If no previous work in the state of the art exists, a clear answer to this should be provided at the architecture level. At the implementation level, it should be clear that although a definite full-fledged solution is perhaps out of the scope of the project, it is very desirable that the prototype takes into account the requirements and doesn't break any cornerstone principle.

At the current state of the project, for the third group, the two most security-relevant topics brought up by UNIFY are the management of the resource allocation process in the RO and the operations on both SG and NF-FG across the different reference points. To address the former, resource policy enforcement has already been discussed in Sec. 3.2.8; whereas for the later, the principles for secure graph operation are outlined next.

As detailed in Sec. 3.1, the UNIFY framework clearly defines its components and reference points. The layer definition in the overarching architecture is oriented to match business boundaries, so the most relevant reference points for secure graph operation would be those between layers: U-Sl, Sl-Or, Ca-Co and Cf-Or. The Ca-Co reference point captures the various interfaces to the north of the underlying controllers and will, most probably, rely on a different entity other than the NF-FG. However, it should include the necessary security functionalities to provide an equivalent behaviour to the rest of reference points.

Keeping the approach selected for the resource policies, the same elements will be used to define the functionalities to be performed by each component of the UNIFY framework. We consider three aspects of graph operation policies with regards to the mapping of policy functionalities to the UNIFY components:

Identities Policies will define the operations allowed for a certain identity (could be a UNIFY User, a Control NF, etc.), so each request must be linked to the correspondent identity to be able to evaluate the policy. For the requests traversing different layers (e.g., a request related to a SG from a UNIFY user to the SL via the U-Sl triggers a request related to a NF-FG from the SL to the OL via the Sl-Or) depending on the implementation scenario the original identity could be maintained through the different requests or be updated in each hop.

Operations Policies will define the operations allowed to each identity, e.g., to be able to differentiate between the functionality exposed to the different UNIFY users envisioned.

Graphs Policies will define the criteria to establish the subset of graphs each specific policy applies to (e.g., operations allowed only in the graphs created by the requester, only specific graphs allowed for deployment, etc.).

The mapping of the PAP follows the same reasoning as for resource management policies and would be part of the OSS, which holds management information; has a complete, domain-wise view; and eases the integration with accounting and back-office processes.

The enforcement of the policy would be done in the U-Sl, Sl-Or and Cf-Or inter-layer reference points, so each component at the south of this reference points would include a PEP. In the RO, it would again be placed in the virtualizers, as they interface with the consumers.

To avoid problems with multiple evaluations of the policies and the complexity from distributed decisions, there should be only one PDP per layer. At the current level of definition of the UNIFY architecture there is no strong case about the placement of the PDP which could be implementation dependent. Fig. 17 shows a possible placement of the PDP in the OSS of each layer.

Considering the aspects described in the graph operation policies, at least the following PIPs would be required: one related to identity, located in the OSS for similar reasons as for the PAP placement; and others related to graph information, located in the SL for SG information and the OL for NF-FG information. Policies including others aspects could require additional PIPs.

Fig. 17 shows one possible mapping of the secure graph operation policy functionalities to the components of the UNIFY architecture.

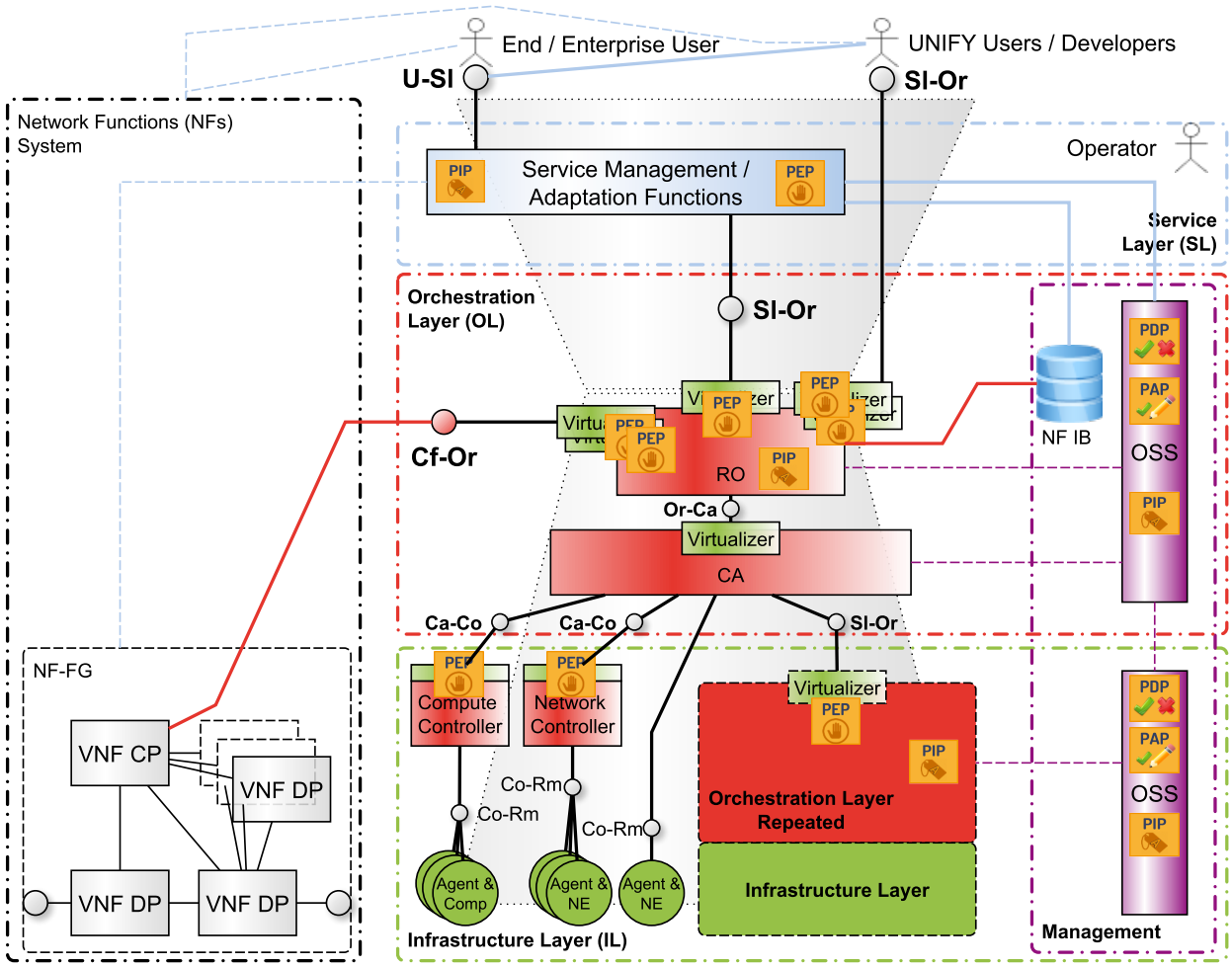

Figure 17: Graph policy functionalities in UNIFY

4 Functional Architecture

The previous section has given a high-level, overarching view on UNIFY architecture introducing the main concepts and components. The next step towards the realization is the functional design of that architecture with lower level details and design decisions. First, we revisit the main functional components and formalize the service primitives associated with the reference points in between. Second, we break down the Service Layer (SL) into detailed functional components. Finally, we break down the Orchestration Layer (OL) into two sublayers and further components. As the UNIFY architecture relies on available controller platforms with corresponding legacy infrastructure elements, and the UN design is detailed in [D5.1; D5.2] the Infrastructure Layer (IL) details are not discussed herein.

The top-level functional architecture is presented in Fig. 18. At the highest level, the SL provides service manage-

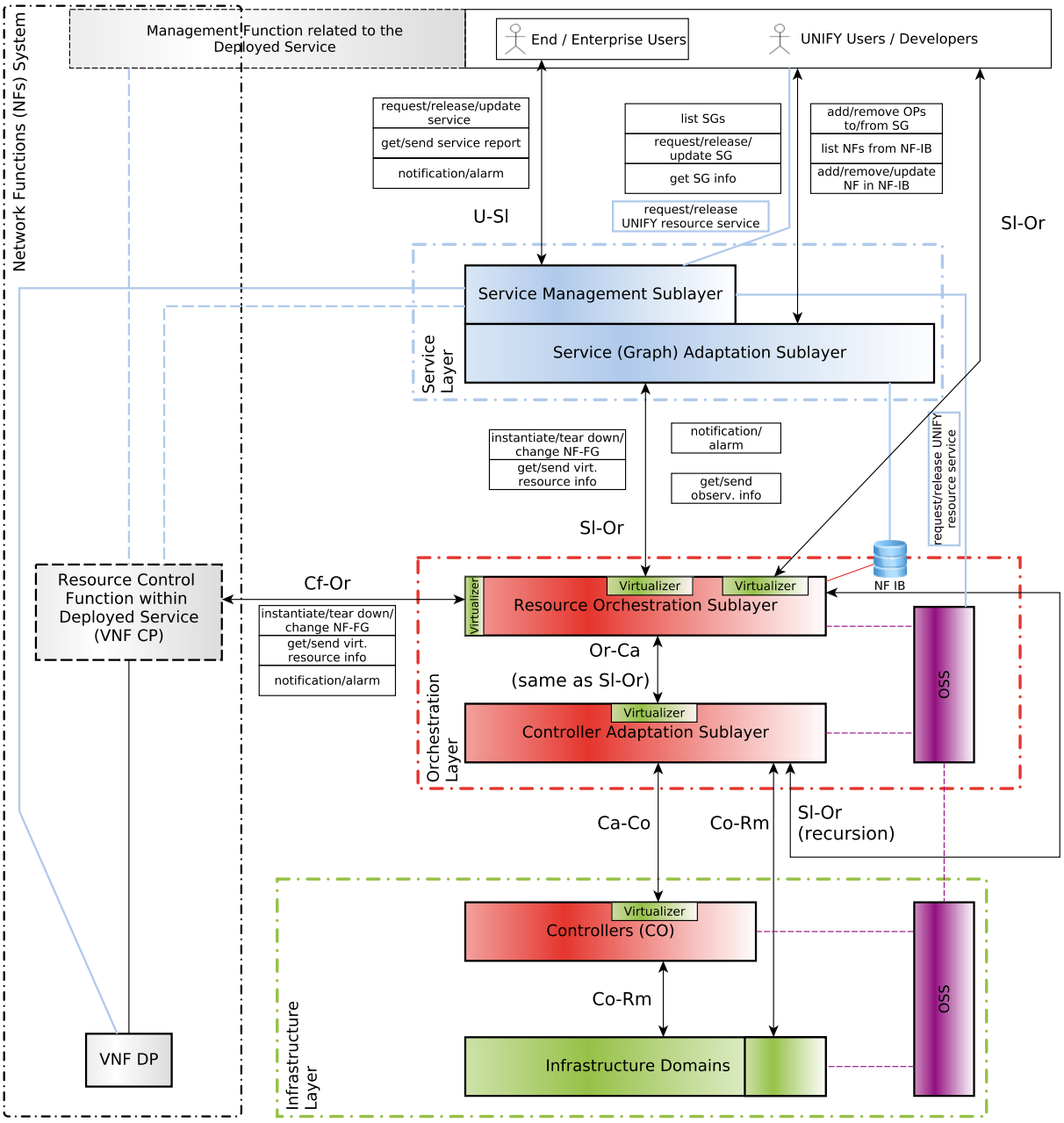

Figure 18: Top-level UNIFY Functional Architecture

ment and service abstraction functions for all types of users (End Users, UNIFY Users, etc.). The components realizing these services are grouped into the Service Management Sublayer. SL can also have management connections to different NFs withing the NFS. UNIFY Users and Developers have special roles, which require special purpose functions and lower level access to the services of this layer. Thus, we introduce a new Service (Graph) Adaptation Sublayer in order to group these lower level functional components.

The OL is decomposed into two sublayers: the Resource Orchestration Sublayer (ROS) and the Controller Adaptation Sublayer (CAS) encompass the functional components corresponding to RO and CA respectively. Both sublayers are managed from a dedicated OSS, while connections to NFs are realized via the Cf-Or reference point.

The main functional elements of the IL are Controllers with their respective infrastructure domains and OSSs.

4.1 Abstract interfaces and primitives

The interoperation between the main components of the architecture (see Sec. 3.2) are realized via the reference points (see Sec. 3.1.2). In what follows, we introduce abstract interfaces with the primitives for each UNIFY reference points. We use the term abstract interface to discussing these service primitives with later protocol mappings, i.e., the abstract interface primitives defined herein can be implemented by different protocols in later phases of the project.

4.1.1 Interface at the U-Sl reference point

The primitives at the U-Sl reference point has to support the following operations:

- **request / release / update service**
  a service (e.g., chosen in the GUI of a management system) is set up / released / updated by the SL (with the help of lower layers) as it is requested by an End or Enterprise User and the status of the operation is sent back to the user as an answer.
- **get / send service report**
  SL provides high-level measurement reports related to the SLA
- **notification/alarm**
  SL sends notifications to the user in case of failure or violation of the SLA.
- **list SG**
  SL lists existing SGs of a given UNIFY User or Developer.
- **request / release / update SG**
  a service described by a SG is started / released / updated by the SL (with the help of lower layers) and the status of the operation is sent back to a UNIFY User or Developer.
- **get / send SG info**
  different types of information on a queried SG is provided by the SL to UNIFY Users.
- **list NFs from NF-IB**
  SL provides the list of available NFs from NF-IB (this could be seen as capabilities).
- **add / remove / update NF in NF-IB**
  NF can be added to / removed from / updated in the NF-IB catalog by the SL according to the request coming from a UNIFY Developer (or User if the user is allowed to do this).

- **add / remove OP to/from SG**
  SL adds / removes OP to/from a given SG according to a Developer's request.
- **request / release UNIFY Resource Service (see Def. 5)**
  to request or release a virtualizer (see Sec. 3.2.1) associated with the UNIFY User or Developer; this is a management request related to the initialization of the UNIFY service. The request is executed through the OSS/BSS in the SL together with the OSS in the OL.

4.1.2 Interface at the Sl-Or reference point

The primitives at the Sl-Or reference point has to support the following operations:

- **instantiate / tear down / change NF-FG**
  ROS takes the NF-FG request, tries to orchestrate it according to the UNIFY Resource Service (see Def. 5 on page 19) and sends back the result.
- **get / send virtual resource info**
  ROS provides resources, capabilities and topology information at compute, storage and networking abstraction (e.g., BiS-BiS resource view).
- **notification/alarm**
  OL sends notification to the SL in case of failure or any violation of KPI thresholds contained in the NF-FG requests.
- **get / send observability info**
  OL provides observability info to the SL.

4.1.3 Interface at the Or-Ca reference point

The CA presents an abstract resource view of the domain to RO and accepts resource requests described by NF-FG, we need similar interactions as it was discussed for the Sl-Or. Thus, the primitives at Or-Ca are the same as at Sl-Or.

4.1.4 Interface at the Cf-Or reference point

The main goal of the Cf-Or reference point is to share the resource management of elastic services between RO and NFs (typically in accordance with a control and data plane split as discussed in Sec. 3.3.2 and 3.3.3). This requires the same primitives as we defined at the Sl-Or reference point. Additionally, the RO may outsource the NF decomposition and NF scaling tasks to an NF running in the NFS. This feature is further elaborated and discussed in [D3.1].

4.1.5 Interface at the Ca-Co reference point

The Ca-Co reference point captures the various interfaces to the North of the underlying controllers, which determine the primitives applied here. We reuse available interfaces here without formulating any requirements. In [D3.1] we discuss some of the related issues.

4.1.6 Interface at the Co-Rm reference point

The interface at the Co-Rm reference point is determined by the protocols used at the southbound interface of the controllers. The definition of this interface is out of the scope of the UNIFY project. In [D3.1] we discuss some of the related issues.

4.2 Service Layer

The SL and its sublayers can be decomposed into finer functional blocks. The functional architecture at the next granularity level is given in Fig. 19.

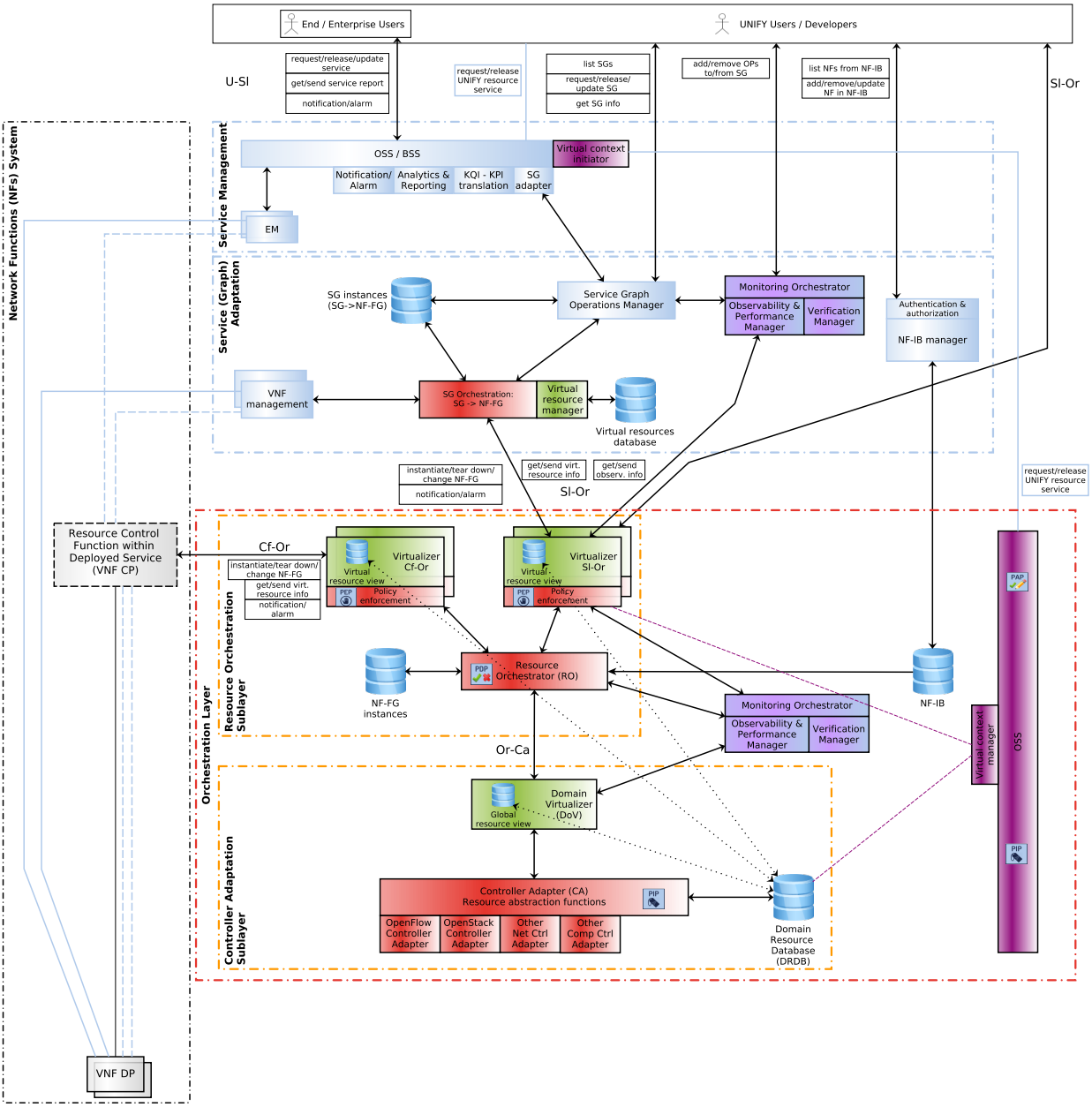

Figure 19: Functional architecture of the Service Layer (SL) and Orchestration Layer (OL)

The Service Management Sublayer encompasses the OSS/BSS and EMs of the SP. We assume that these components are service aware and are readily available from incumbent operations. Additional, virtualization specific, compo-

nents must be added to the management systems, e.g., SG adapter or a virtual context initiator (see later). Services can be requested via the OSS/BSS using, for example, a GUI. The GUI or the service portal can also be used for presenting different types of reports on the service back to the users. The OSS/BSS contains several modules regarding reporting, notification mechanisms, translation of KQI to KPI parameters, SG adaptation, etc. The SG adaptation is responsible for transforming the high-level service into a SG. The OSS/BSS implements the primitives regarding service operations (request, release, update), service reports and notifications as it is shown in Fig. 19.

EMs are service aware configuration and management components connecting to NFs running in the NFS through their traditional management interfaces (e.g., SNMP). Typically, we have one EM instance per NF Functional Type (e.g., firewall).

The Virtual context initiator handles the creation of new Virtualizer components at the RO via the OSS of the OL layer. On the one hand, this event can be triggered by UNIFY Users or Developers calling a dedicated function, e.g., request UNIFY Resource Service. On the other hand, the OSS/BSS in the SL can request Virtualizers during for some of its Users, e.g., for a Developer or a power user directly controlling resources through NF-FG requests.

The Service (Graph) Adaptation Sublayer contains the virtualization related components of the SL. UNIFY Users and Developers may requests direct access to these elements. Operations with SGs, such as listing active SGs, starting, stopping or updating services described by SG, querying information on given SG can be called directly via the Service Graph Operations Manager. Typically, there is a single Service Graph Operations Manager in the SL and it connects to a single Virtualizer of the ROS. The resources provided by the Virtualizer (via "get/send virt. resource info" primitives) are stored in Virtual resources database. The SG Orchestration module is responsible for mapping an SG request into an appropriate NF-FG as described in Sec. 3.2.3, e.g., into a BiS-BiS virtualization stored in the Virtual resources database. (A helper object is also introduced to handle operations corresponding to virtual resources called Virtual resource manager.) The resulted NF-FG is sent down to the RO via the Sl-Or reference point using the "instantiate/tear down/change NF-FG" primitive provided by the corresponding Virtualizer object. This mapping information (SG to NF-FG) is also stored in a database containing active SG instances in the SL. The Virtualized Network Function Manager (VNFM) module is responsible for the lifecycle management of VNF instances (e.g., start/stop instances, dump/reload configurations). This component performs service agnostic tasks typically related to the operation of a VNF in a virtualized infrastructure like refreshing software versions, pausing/resuming, dumping/initiating states, etc. Similar functionality can be optionally delegated to a Resource Control Function as a deployed NF running in the NFS (see Sec. 3.3.3).

The Service (Graph) Adaptation Sublayer has additional functional elements for accessing the NF-IB. Using the NF-IB manager, Developers or UNIFY Users are able to query available NFs and add, remove or update information on NFs in the database. The NF-IB is used by the RO during orchestration, therefore it is indicated as part of the OL in Fig. 19.

Another important components of the Service (Graph) Adaptation Sublayer realize SP-DevOps services [D4.1]. The Monitoring Orchestrator maps monitoring requirements coming from the SG or from the UNIFY Users/Developers to available monitoring services and observability points. The Observability & Performance Manager can collect Observability information from lower layers after configuring them via dedicated interfaces. Furthermore, Developers or UNIFY Users can directly add OPs to a given SGs by the "add/remove OPs to/from SG" primitives. These mechanisms require additional primitives and will be discussed in more details in [D4.2]. The Verification Manager is able to conduct verification processes on different tasks performed by the SL components, e.g., verification of the mapping of an SG to an NF-FG at a virtualized resource.

4.3 Orchestration Layer

The decomposed view of the OL is shown in the lower part of Fig. 19. Two logical groups, referred as sublayers, are formed around the RO and the CA. The encompassed modules support the operations realized by these two main components. Additional blocks support SP-DevOps processes or provide management functions related to this layer. It is worth noting that the purple OSS shown in the right hand side is different from the OSS/BSS operating at the SL. The OSS in the OL is managed by the OSS/BSS in the SL and its role and responsibility is to manage and operate the RO and the CA.

The CAS, which is the lower part of the OL, creates a global (abstract) domain view hiding the details of the underlying technologies, controller platforms and infrastructure components. In order to create and manage this virtual view, the main component (CA) interacts with different types of network and compute controllers and collects virtualized or real infrastructure resources into a database referred as Domain Resource Database (DRDB). A number of adapters can be integrated with the CA to implement interfaces towards different types of controllers, e.g., OpenFlow Controller Adapter, OpenStack Controller Adapter. The main role of the CA is to provide resource abstraction functions for the Domain Virtualizer (DoV) module. The DoV creates a domain wide, unified global resource view of all underlying resources and exposes that to the RO through the Or-Ca interface. The global resource view managed by the DoV can be considered as a simplified view of DRDB[2]. As a result, the RO and its orchestrating algorithms are infrastructure agnostic. The idea behind the RO and CA split is to separate the concerns: orchestrating over abstract resources (RO) and mapping the execution into different technologies, domains and protocols.

The ROS is the upper part of the OL. Its main role is mapping the NF-FG request coming from the Sl-Or or Cf-Or reference point to domain global resources as shown by the CA. The mapping algorithms of the RO may adhere to different operational objectives, like minimizing costs, energy; maximizing utilization, etc. The ROS contains the RO, Virtualizers with Policy enforcement functions (PEP), and an NF-FG instances database. A Virtual context manager, highlighted from the OSS, is responsible for creating and configuring consumer specific Virtualizers with PEPs via the "request/release UNIFY Resource Service" primitives. Consumers of the Virtualizers are UNIFY Users or NFs in the NFS (see Sec. 3.2.1 and 3.3.3). The virtualization is performed by the Virtualizer component by presenting a consumer specific DRDB. This view is provided to the consumers via the "get/send virt. resource info" primitives. An NF-FG request via the "instantiate NF-FG" primitive from the consumer is mapped to the global domain resources by the RO and stored in the NF-FG instances database. The mapping is performed based on information stored in the NF-IB, which database is managed from the SL (see NF-IB manager). The NF-FG instances database stores the original request and the NF-IB entries used to decompose or map it. The mapping to the resources are stored in the DRDB. Virtualizers are connected to the RO through their policy enforcement module (PEP). Resource policies related to the service contract must be enforced here related to compute, storage and networking abstractions (resource specific and service agnostic view). For example, a PEP could limit the available resources (max 10 virtual CPU, max 5 GByte RAM, less than 100Mbps in/out rate, maximum cost of resources, etc.) or the capabilities as available NF types. Such policy information could be extracted from the original service request and/or from consumer policies (e.g., maximum cost, etc.) and might involve additional services, e.g., calculating the resource cost of a service.

The Monitoring Orchestrator is responsible for the allocation and operation of monitoring resources by translating monitoring related NF-FG requests into monitoring functions and observability points and preparation of the instantiation of the selected elements. The result is forwarded to the DoV or controller in the infrastructure layer. The Observability & Performance Manager together with the Verification Manager are responsible for SP-DevOps related tasks in the OL. They are connected to similar functional blocks of the SL via the corresponding Policy enforcement module

[2] This relation is indicated by dotted lines between the databases in Fig. 19.

and Virtualizer. By this means, the lower level components can be configured and different types of observability information can be reported upwards. These modules also interact with the RO directly in order to provide performance information which can be taken into account during the orchestration process. To collect these measurements from the lower layers, another direct connection to the DoV is necessary.

5 Towards an Integrated Prototype: Aspects of the System Architecture

In the execution of the UNIFY project we follow agile development processes. In parallel to the definition of the overarching and functional architecture, we build limited functionality proof of concept prototypes for validations and learnings about our ideas. This section is devoted to briefly summarize some of these preliminary implementation efforts paving the way for a complete system architecture and integrated prototype.

Since with UNIFY we aim to integrate multiple technologies into the compute, storage and network abstraction, we have started to work with different types of infrastructure to understand if our abstraction (generalization) is applicable or not. There different prototyping environments correspond to the UN, legacy DC solutions and light-weight, emulated networking frameworks. We will briefly describe these in this section.

5.1 ESCAPE prototyping framework

Mininet [LHM] is a light-weight network emulation tool enabling rapid prototyping. It is a proper candidate to build a UNIFY development framework around that in order to make agile prototyping possible. Therefore, we have established such a framework including all layers of UNIFY architecture (Infrastructure Layer (IL), Orchestration Layer (OL), Service Layer (SL)) and demonstrated in [Cso+14b] under the title of "Extensible Service ChAin Prototyping Environment using Mininet, Click, NETCONF and POX (ESCAPE)".

The system architecture of ESCAPE is shown in Fig. 20.

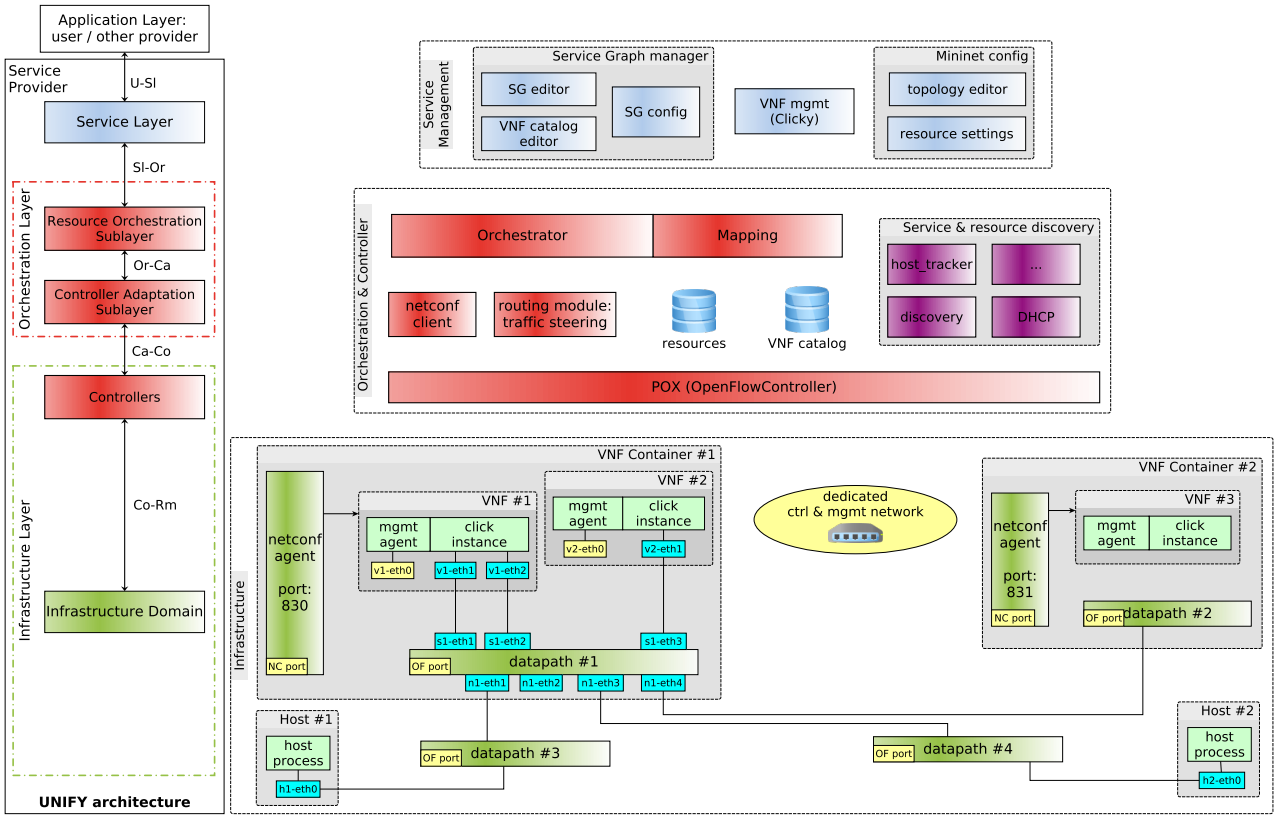

Figure 20: The system architecture of ESCAPE with the corresponding UNIFY layers

The main goal of ESCAPE is to support the development of several parts of the service chaining architecture. On the one hand, the framework fosters VNF development by providing a simple, Mininet-based environment where SGs,

built from given VNFs, can be instantiated and tested automatically. Here, VNFs are implemented in Click modular router [Koh+00] and run as distinct processes with configurable isolation models (based on Linux cgroups), while the infrastructure consists of OpenFlow switches (Open vSwitch). A dedicated controller application (implemented in the Python-based software-defined networking (POX) [POX] OpenFlow controller platform) is responsible for steering traffic between VNFs. On the other hand, the framework supports the development and testing of orchestration components. Mininet was extended by NETCONF capability in order to support managed nodes (VNF containers) hosting VNFs. The orchestrator also communicates the NETCONF protocol and it is able to start/stop VNFs on demand. The paths are handled similarly by our POX controller. On top of these, a MiniEdit based Graphical User Interface (GUI) can be used to describe SGs (with requirements) and test topologies (resources in the network). Then, given SGs are mapped to available resources dynamically, and lower-layer orchestration methods are invoked.

5.1.1 Infrastructure Layer

In ESCAPE, a light-weight, Mininet-based infrastructure emulates the lowest layer of UNIFY architecture. The goal of this approach is twofold. On the one hand, this simple environment makes faster development possible at the higher layers (especially in the OL) as it can be done in parallel with the UN design and prototyping. On the other hand, several elements can be used later in an integrated framework. For example, VNF implementations can easily be migrated to UN execution environment, the experiences with the implementation of the abstract interfaces can result in useful feedback or refinement of the interface details.

Mininet had to be extended in several directions in order to fit UNIFY architecture requirements. The main components of the network infrastructure of ESCAPE are the following (see the bottom part of Fig. 20): **VNF Containers** (or Execution Environments or Nodes), **OpenFlow switches** (OVS instances), **virtual Ethernet links** (veth pairs), **end hosts**, **VNFs**.

Mininet was extended with the notion of VNF and VNF Container. Here, VNF Container is a bash process with configurable isolation models (based on Linux cgroups) with a limited amount of CPU, memory, etc. resources assigned to it. VNF is a Click (modular router) process running in a VNF Container with configurable isolation models (similarly based on Linux cgroups). It has multiple datapath interfaces connected to OVS ports, and additionally has a dedicated control and management network connection (see mgmt agents of VNFs).

In order to support remote management of VNF Containers, we have extended Mininet with NETCONF capability. Each VNF Container includes an OpenFlow switch and additionally a NETCONF agent which is responsible for starting/stopping VNFs and connecting/disconnecting VNFs to/from OpenFlow switches. The NETCONF implementation is based on the open source OpenYuma tool which was extended to support multiple instances on single machines with given ports.

The NETCONF agents are controlled from the Orchestration layer by a NETCONF client component. The used interface is a realization of the Co-Rm reference point in UNIFY architecture. The interface with the remote procedure calls and data structures is described by a Yang model constructed for this special purpose. Based on the Yang model, low-level instrumentation codes were implemented as NETCONF modules to hide the infrastructure level details. It is worth noting that this approach supports migration later e.g., to UN or to OpenStack DC environment (only the instrumentation codes have to be replaced).

5.1.2 Orchestration Layer

Implementing the OL, we aimed at constructing a modular framework where several components can be dynamically changed as going forward with prototyping. During this phase, we implemented simple algorithms with limited capabilities as proof of concepts only.

In the current version of ESCAPE we have combined OL and controllers into a common layer which is built on POX (see the center part of Fig. 20). This approach was suitable to follow an agile prototyping way.

POX implements the Co-Rm interface for OpenFlow domains. On top of that, we have built a routing module implementing the traffic steering between deployed VNFs. This module is responsible for gaining the forwarding information from an NF-FG and calling appropriate POX functions to send flow entries to OpenFow switches. As it is implemented internally in POX, the communication via the northbound interface is not necessary. This can be considered as a simplified implementation of the Ca-Co reference point. Computing elements, i.e., VNF Containers are managed via the NETCONF protocol. For this purpose, we have integrated a NETCONF client with the framework to communicate with the peer agents running at the VNF Containers. The clients can call the exposed RPCs defined by the previously mentioned Yang model. This NETCONF client is a Controller Adapter module for computing resources.

The main component of this layer is the Orchestrator and its mapping module. These components implement the RO of UNIFY architecture. The mapping module is responsible for mapping NF-FG to available resources where the optimization algorithm can easily be changed. Here, we used a simplified, preliminary version of NF-FG with limited set of information. The Or-Ca reference point is internal in this approach.

Additionally, we have other POX applications to support automatic configuration and management related tasks.

A first, simplified version of the NF-IB was also implemented referred as VNF catalog which contains a built-in set of useful VNFs. This is a SQL-based (sqlite3) database integrated with POX and it contains only the most relevant parameters of the VNFs, such as name, type, description, Click script, dependency information. Click scripts are stored by Jinja2 based templates yielding easy and flexible parameterization. Furthermore, the designed VNF catalog provides interfaces to add, remove or change VNFs in the catalog during run time.

5.1.3 Service Layer

The SL of ESCAPE with the GUI is based on Miniedit as it is shown in the top part of Fig. 20. It contains a SG manager which is capable of describing/configuring/editing SGs, configuring requirements (SLAs), such as delay, bandwidth, on given sub-graphs. This SG is converted to an NF-FG which is sent to the Orchestrator via an internal interface which is a first and simplified implementation of the Sl-Or interface and the corresponding primitives.

A Mininet configuration component gives a graphical front-end to the Mininet-based infrastructure. Here, we can describe physical topology containing OpenFlow switches (e.g. OVS), VNF Containers, Service Attachment Points (SAP) (currently implemented as hosts) and links. Then the emulated network environment will be started according to these configuration parameters. Moreover, resources of the emulated infrastructure such as, CPU fraction, memory fraction, link bandwidth, link delay, can also be configured.

For VNF management and visualization purposes, we use Clicky. Clicky is a GUI for Click modular router which can be used for configuring and monitoring Click instances.

5.2 OS/ODL based infrastructure

We have combined the prototyping framework presented in Sec. 5.1 with an OpenStack (OS) data center with the OpenDaylight (ODL) controller to form a multi-domain setup where multi-level orchestration could be performed. The main

components of this setup are shown in Fig. 21, as presented in [Cso+14a]. In Fig. 21 the main components are the GUI, the global orchestrator and the Mininet and OS/ODL domains, the latter with a local orchestrator. The orchestration spans through multiple domains and service graphs can be instantiated by third parties. For this reason, we developed communication interfaces and an abstract view of a whole domain which can be adopted by a global orchestrator to use the resources in a separate domain through its local orchestrator. Our framework currently supports Mininet-based virtual domains and data centers managed by OS/ODL, denoted by OS/ODL domain. In this sense the OS/ODL domain is something similar to a UN. VNFs are implemented in Click and in case of a Mininet domain, they run as distinct processes with configurable isolation models, while in OS/ODL domain, virtual machines are deployed to run Click processes. We have a VNF catalog storing available and deployable network functions. The infrastructure comprises OpenFlow switches and VNF containers (managed nodes) hosting VNFs, while a dedicated controller application (implemented in POX) is responsible for traffic steering.

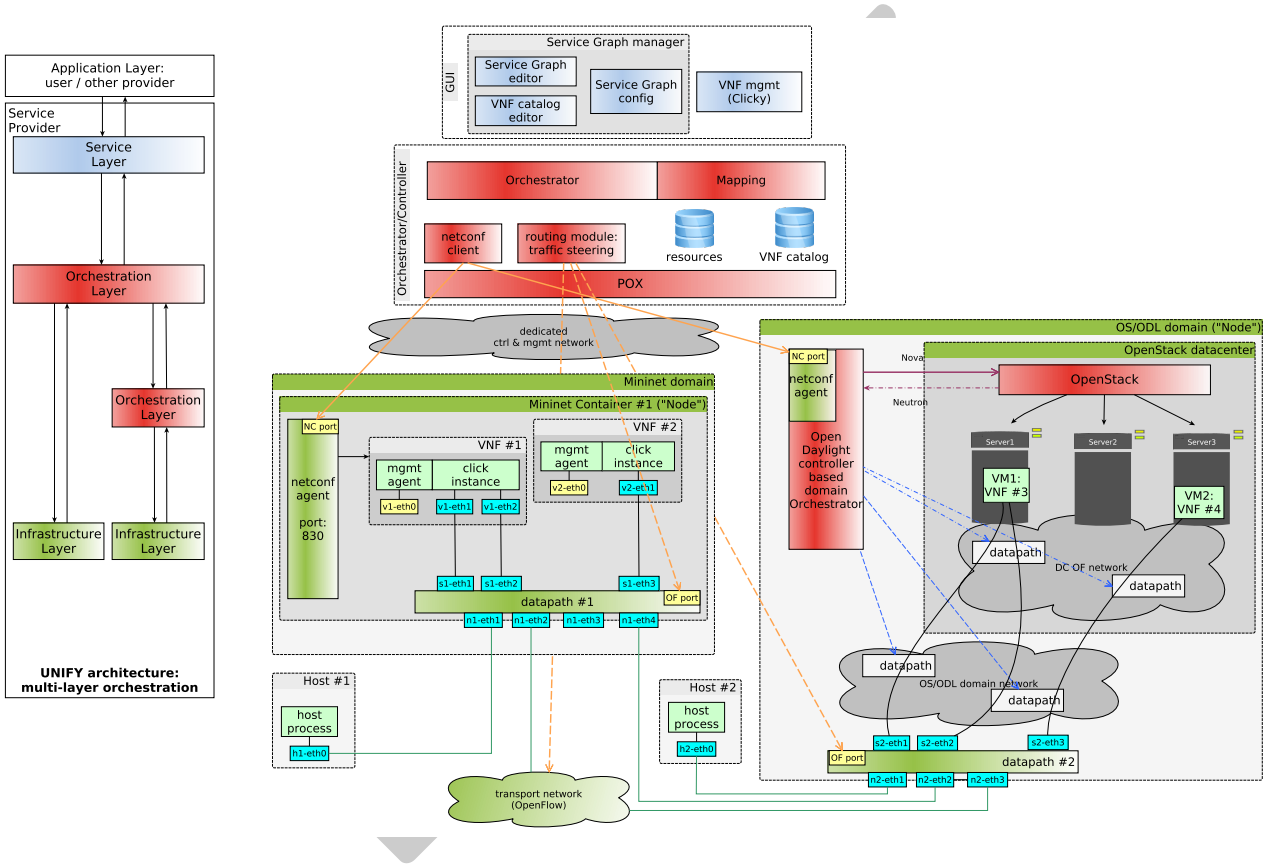

Figure 21: Logical view of the OS/ODL setup

Our aim was to provide an abstract view of our OS/ODL domain for the ESCAPE orchestrator. We expose the whole domain as a simple node with the capability of service graph instantiation which is managed solely by the domain's local orchestrator. By the global orchestrator of ESCAPE, this domain is treated as another VNF container. The global orchestrator has a global view of the available resources and the capabilities of each node running in its domain. It is capable of partitioning a service graph into multiple subgraphs which can be given to the OS/ODL domain for further decomposition and instantiation. In the simplest case the subgraph could comprise only one network function which would be instantiated in the OS/ODL domain's data center. After successful instantiation of the subgraph, the OS/ODL domain's local orchestrator notifies the global orchestrator and provides information on access to the newly created network function.

Similarly to Mininet containers, OS/ODL domain has two control interfaces implemented by dedicated components.

The first one is an OVS switch steering traffic between the edge of the domain and specific ports on which the instantiated network functions are reachable. This interface corresponds to a Co-Rm interface. The second one is a NETCONF agent module which is implemented in the ODL controller and tightly integrated with its framework, corresponding to an Sl-Or interface. The plugin calls the REST API of OS ("Nova") to request a network function which is then initiated in the data center as a virtual machine. After the network function is booted up and the relevant network configurations are deployed in the data center's network, the ODL controller sets up an overlay topology in the domain's network and creates a new port in the OVS switch which is directly connected to the overlay. This port's id is signaled back to the global orchestrator through the NETCONF protocol. Note that the edge domain OVS switch has only one master controller which is the global orchestrator. The ODL controller has the right to create ports on this switch but nothing more. In our implementation the OS cloud's internal networking is controlled by the same ODL controller which is responsible for the domain network. OS makes network configuration requests to ODL via it's "Neutron" REST API. Then ODL configures the DC's internal datapath elements via OpenFlow.

With the setup, a complex service described by a service graph (with given requirements) can be requested via the service layer's global orchestrator's GUI. This complex service will be decomposed by the orchestrator to smaller blocks. Some of these blocks will be instantiated in Mininet containers, while others in the OS/ODL cloud which makes further local orchestration. This decomposition and instantiation details are hidden from the requester of the high level complex service.

5.3 Universal Node (UN) prototype

This section describes the implementation of UN consisting of a single physical server, which implements the UN architecture depicted in Fig. 7.

The main external interface of the UN is the northbound API, which has been implemented as a REST interface accepting an NF-FG from the upper layer orchestrator. This data contains all the information needed to deploy the graph on the physical node, namely the VNFs and the connections among them. In the case of the implemented prototype, whose architecture is depicted in Fig. 22, a prototype service logic has been developed as well, targeting the "infrastructure virtualization" use case, in order to better demonstrate the prototype in a realistic use case. However, as evident from the picture, the above service module is seen by the UN as an upper layer orchestrator, given its output in the NF-FG form.

The control logic of the UN is split in the following components, according to the UN architecture:

- an orchestration layer, which implements the northbound interface and that can host scheduling algorithms in order to possibly optimize the VNF placement within the UN itself (not yet implemented the time of writing this deliverable);
- two infrastructure controllers, respectively for the compute and networking component:
  - the compute controller handles the lifecycle of the VNF, such as starting/stopping the physical compute instance. It is further split in three logical modules dedicated to the three execution environments that are currently supported, namely DPDK processes, Docker containers, and traditional virtual machines.
  - The network controller translates NF-FG network commands into a set of OpenFlow flowmods that are passed to the proper OpenFlow controller and that are used to actually configure the switching infrastructure.

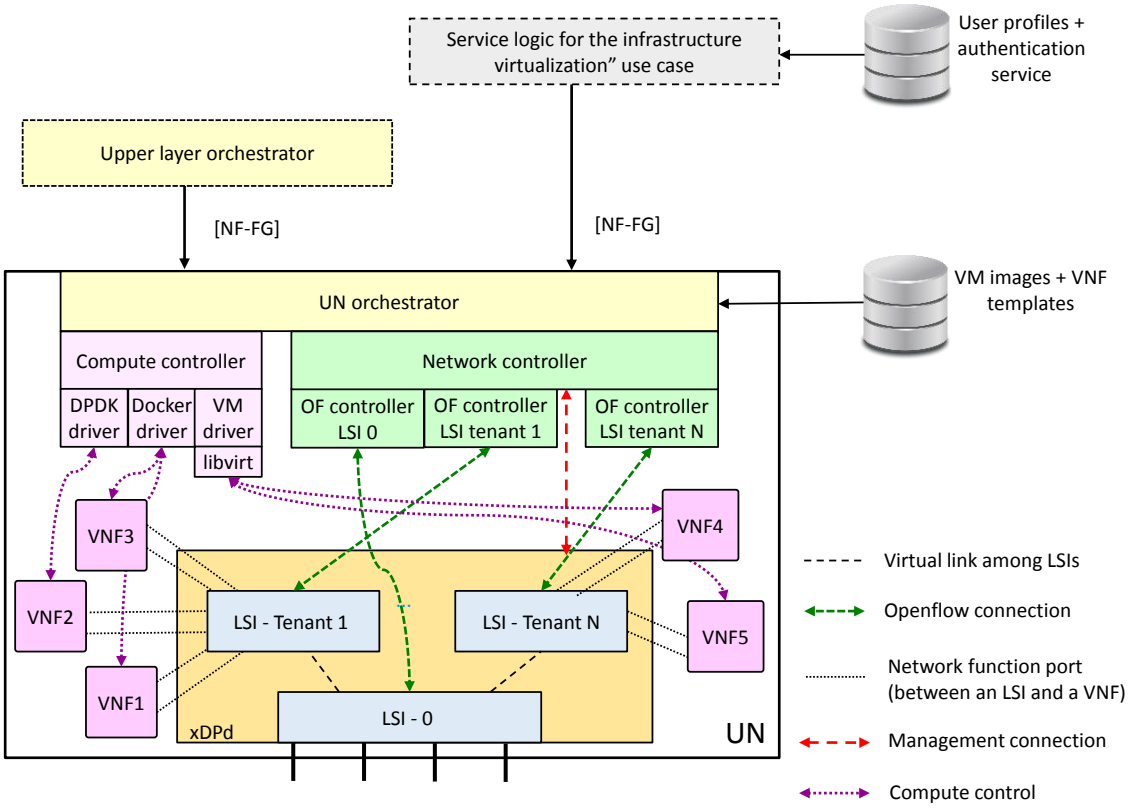

Figure 22: Internal architecture of the Universal Node (UN) prototype

The traffic steering among the different components is based on the eXtensible OpenFlow DataPath daemon (xDPd) softswitch a framework that supports the dynamic creation of several (pure) OpenFlow switches called Logical Switch Instance (LSI); each LSI can be connected to physical interfaces of the node, to VNFs, and to other LSIs. In the prototype, a different LSI (called tenant-LSI) is dedicated to steer the traffic among the VNFs of a specific NF-FG, while the LSI-0 is in charge of classifying the traffic coming from the network (or from other graphs) and of delivering it to the proper graph implementation. It is worth noting that the LSI-0 is the only one allowed to access to the physical interfaces, and that the traffic flowing from one graph to another has to transit through the LSI-0 as well.

LSIs access to the network ports through the Data Plane Development Kit (DPDK) framework: the `igb driver` is used in case of physical interfaces; the `kni driver` and the `rte_rings` are instead used to exchange packets with the VNFs, according to the type of the VNFs themselves (i.e., VMs, Docker containers, or DPDK processes).

When a NF-FG description (either a new one or an update of an existing one) is received by the UN orchestrator, this module:

- replaces the generic VNF endpoints with the actual vNICs created on the server;
- retrieves an implementation for each VNF required and installs it;
- in case of a new NF-FG, it instantiates a new tenant-LSI on xDPd and connects it to the LSI-0 and to the proper VNFs;

- creates an OpenFlow controller that allows to insert forwarding rules into the flow table(s) of the new LSI, in order to steer the traffic among the VNFs as required by the graph.

5.4 Future directions and plans on integration

Now that the final overarching and functional architecture definition is closed, the implementation efforts will continue along three parallel lines:

1. Mapping of the implemented functionality to the final architecture and reference points to identify the possible adaptations needed;
2. Implementation of those capabilities most significant for the UNIFY architecture not already covered;
3. Integration of the diverse implementations and WP-specific prototype towards the Integrated Prototype.

For the first line of work, UNIFY framework has clearly defined the reference points (U-Sl, Sl-Or, Or-Ca, Ca-Co, Co-Rm and Cf-Or, see Sec. 3.1.2 and Fig 1) and information models (SG and NF-FG, see Sec. 3.2.3 and 3.2.2) in the final architecture. This should make the mapping and identification a straightforward task and ease the scheduling and dimensioning of efforts for the next lines of work.

For the second line, each technical WP is already working in different aspects of the UNIFY framework not yet included in prototype activities, such as, for example:

1. supporting recursive and multi-level orchestration, optimized placement, NF decomposition, possibilities of the Cf-Or or integration of NF-FG across different domains in WP3;
2. defining a language to describe monitoring functions and observability points, implementing a multi-layer and multi-component debugging tools and other tools for verification and troubleshooting in WP4;
3. and improving UN with advanced and optimized capabilities, including support for different types of NF and steering mechanisms in WP5.

The third line of work will be carried in task T2.3, which will first focus on aligning individual implementations and to take care that an exchange of information is performed, moving later to define an integration plan towards an integrated prototype.

6 Preliminary Evaluation of the UNIFY Architecture and Elastic Services

For a state of the art review we would like to refer the reader to Sec. 2 of [D2.1]. Therefore, we only revisit here the ONF SDN and the ETSI NFV architectures in comparison with the final UNIFY architecture and some highlighted services.

6.1 Virtualization: SDN and NFV

ONF works on the definition of an SDN architecture[Ope14b]. They focus on three layers: data, control and application plane layers, but also include traditional management plane and end user systems into their architecture. SDN applications are defined as control plane functions operating over the forwarding abstraction offered by the SDN Controller. The applications connect to the SDN controller via the A-CPI reference point. The SDN control plane's main responsibilities are i) creating the domain wide abstraction for internal use; ii) creating application or client SDN controller specific virtualization and policy enforcement; and iii) coordinating resources and operations for virtualization. The data plane of the SDN architecture constitutes of NEs that deals directly with customer traffic. NEs are connected to the SDN Controller through the D-CPI reference point. The right hand side of Fig. 23 shows an illustration of the SDN components and the corresponding reference points.

Since the SDN architecture allows other SDN Controllers (clients) to connect to the north of an SDN Controller via the I-CPI reference point the architecture is recursive. Therefore, automated network orchestration can be executed in multi-level virtualization environments, as long as resource virtualization and client policies related to resource use can

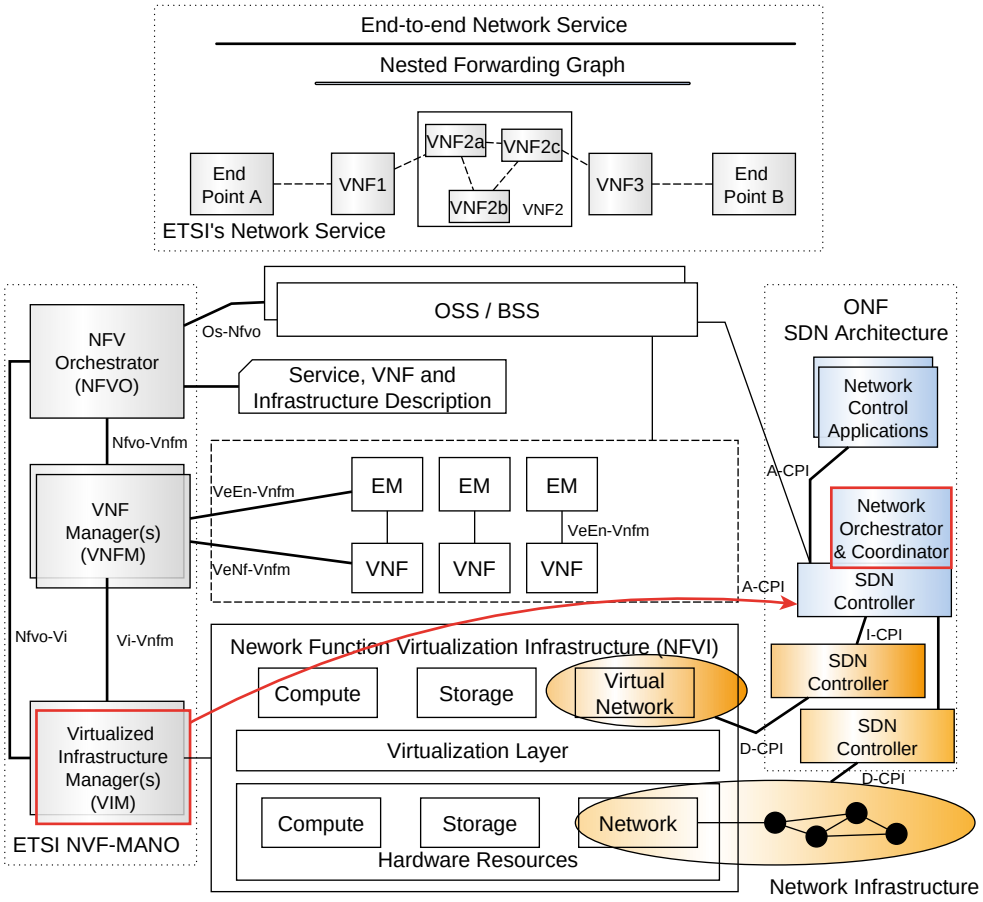

Figure 23: ETSI NFV and ONF SDN architectures side by side

be set. Such recursive automation enables clear separation of roles, responsibilities, information hiding and scalability. It also provides efficient operations in multi-technology, multi-vendor, multi-domain or multi-operator environments.

If we look into the user service aspects, then flexible service definition and creation may start by abstracting and formalizing the service into the concept of NS[3]. In the ETSI NFV framework [ETS13a] the NS is formulated as a Virtualized Network Function Forwarding Graph (VNF-FG) (see top of the Fig. 23). These graphs represent the way in which service end points (e.g., customer's access) are interconnected with the desired NFs, such as firewalls, load balancers, Dynamic Host Configuration Protocol (DHCP) servers, etc. VNF-FG representations form the input for the management and orchestration to instantiate and configure the requested service. There is an ongoing work in the ETSI NFV ISG on a MANO framework [ETS14] for NFV.

MANO components are the Network Function Virtualization Orchestrator (NFVO) for the lifecycle management of the services; VNFMs for lifecycle management of individual VNFs; and Virtualized Infrastructure Managers (VIMs) for controlling and managing compute, storage and network resources (see left hand side of Fig. 23). VNFs are instantiated in a Network Function Virtualization Infrastructure (NFVI) by the VIM through technology specific resource controllers like an SDN Network Controller or a Compute Controller. The MANO framework is completed by connections to traditional management functions. These include EMs connected to the VNFs and OSSs and BSSs . The ETSI NFV architecture document also identifies the reference points between the functional components. The split of roles and responsibilities follow virtualization related tasks: NFVO, VNFM and VIM are responsible for the virtualization related aspect of the management and orchestration.

If we put side-by-side the SDN and the NFV architectures, we can see that the VIM talks to an SDN Controller (or equivalent) to orchestrate the virtualized network in an NFVI (red arrow in Fig. 23). A VIM can use the I-CPI or the A-CPI SDN reference points to control the network. Logically, however, a VIM and an NFVO will perform network resource orchestration similar to a Network Orchestrator and Coordinator within the SDN architecture.

There is, however, at least one major difference between the designs of the NFV and SDN architectures: SDN relies on a basic forwarding abstraction, which can be reused recursively for virtualization of topology and forwarding elements, while the NFV framework offers significantly different services on the top compared to what it consumes at the bottom. Therefore, it is not straightforward, how to offer NFV services in a multi-level hierarchy with the MANO framework.

We believe that with combined abstraction of compute and network resources, we can logically centralize all the resource orchestration related functionalities existing distributively in the MANO framework into a resource orchestration. Such architecture, as proposed by the UNIFY project, can enable automated recursive resource orchestration and domain virtualization similar to the ONF architecture for NFV services.

We show in Sec. 6.2 how ETSI MANO, ONF SDN and UNIFY architectures compare with regards to their main functional components.

We showed that the logical centralization of joint compute and network resource orchestration enables direct control of elastic resources for the network functions. This embedding of resource control within a deployed service resembles a recursive architecture similar to ONF SDN one. We compare in details elasticity control in Sec. 6.3.

6.2 ETSI MANO, ONF SDN and UNIFY

The UNIFY architecture is very similar to the SDN architecture, but it operates at the joint compute and network abstraction in each of its components. Therefore, the OL of UNIFY is equivalent to SDN's Controller and "Network Orchestrator and Coordinator" functionality. The SL components who directly interact with the UNIFY OL can be considered as "SDN

[3] ETSI term, similar to SG in UNIFY

Application plane" components (see middle and right hand side of Fig. 24). The UNIFY OL can be used to construct a multi-level hierarchy of virtualization and control similar to the ONF SDN architecture.

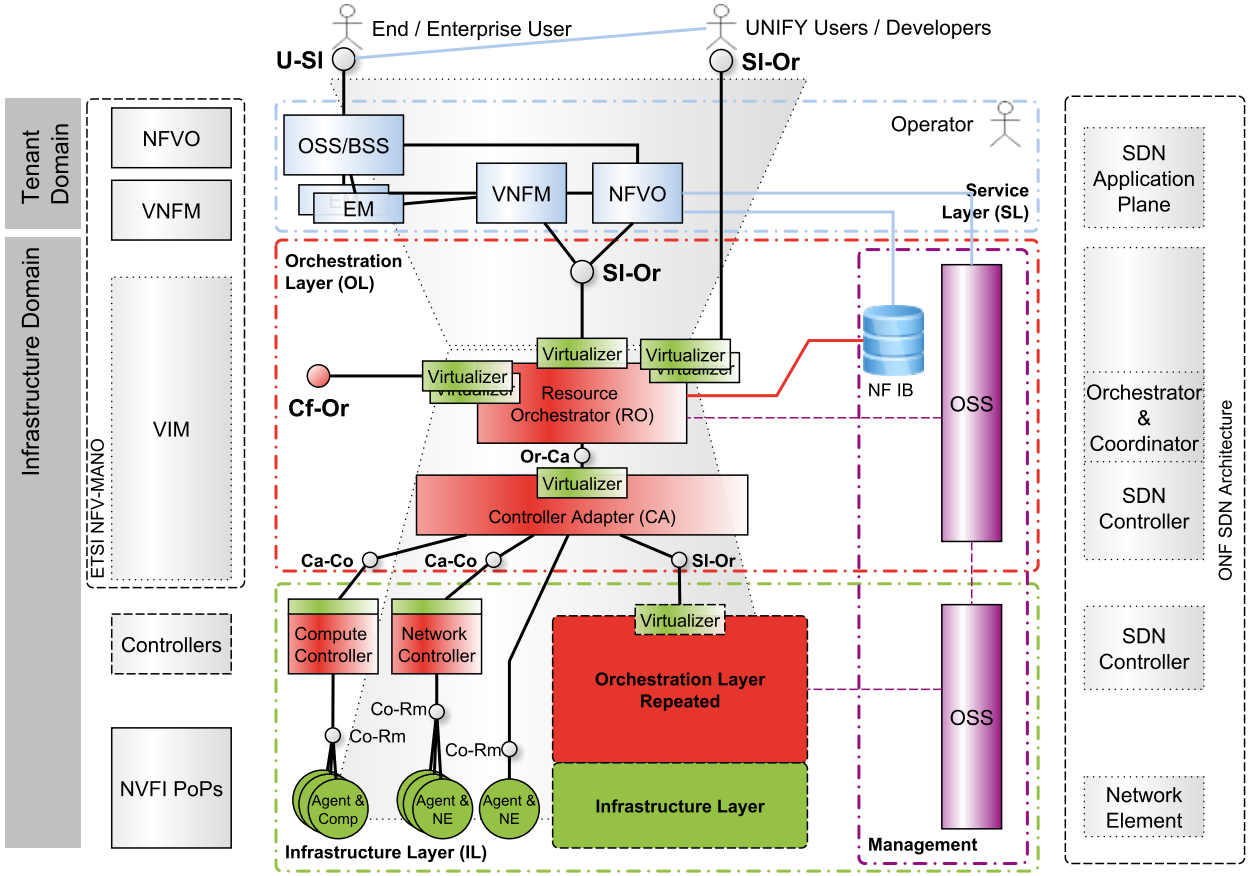

Figure 24: ETSI NFV, ONF SDN and UNIFY architectures side by side

According to the ETSI MANO framework we have to compare NFVO, VNFM and VIM components to their UNIFY correspondent. According to the detailed discussions of the UNIFY Functional Architecture in Sec. 4, we can conclude with the followings (also see left side and the middle of Fig: 24):

VNFM: ETSI's VNFM is responsible for lifecycle management of individual VNF. Equivalent to that fuction exists in the UNIFY SL with the same name.

NFVO: ETSI's NFVO is responsible for the lifecycle management of the services. In UNIFY the SG Orchestrator is responsible to a similar function.

VIM: ETSI's VIMs are responsible for controlling and managing compute, storage and network resources. According to the MANO framework, there can be as many VIMs as many NFVI Point of Presences (PoPs). In UNIFY we logically centralized all resource management function with our joint programmatic reference point at Sl-Or, therefore our OL obsoletes VIMs.

6.3 Elastic services

According to the MANO framework the VNFM is responsible for lifecycle management of individual VNFs. For example, the VNFM function may monitor KPIs of a VNF to determine scaling operations. Scaling may include changing the

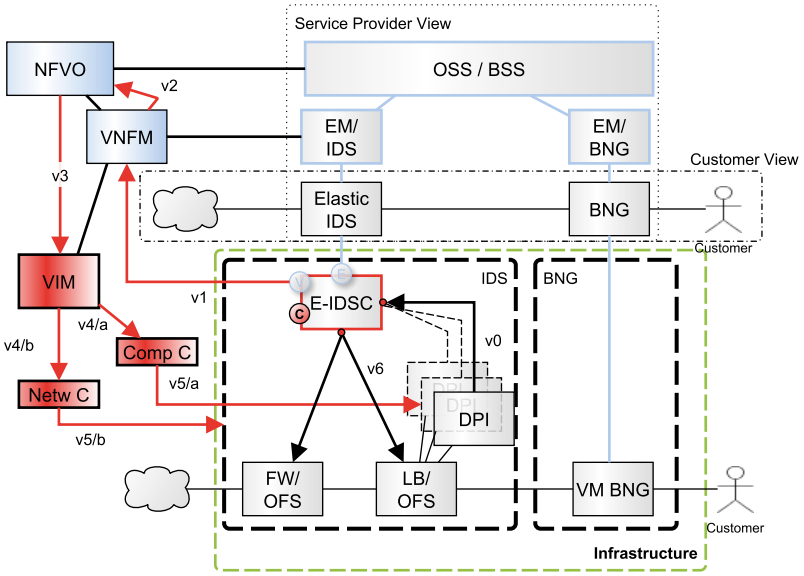

Figure 25: Elastic control loop according to the ETSI MANO framework

configuration of virtualized resources, like adding/removing CPUs, adding/removing VMs and adding/removal of associated network resources. The VNFM functions are assumed to be generic (VNF agnostic) and exposed by an open interface (VeNf-Vnfm) for the VNFs (see Fig. 23). However, beyond resource control, VNFM's other functionalities are VNF instantiation and configuration; updates, upgrades and modifications; collection of VNF related performance measurements and event information; VNF assisted or automated healing; and coordination and adaptation between corresponding the EM and the VIM.

On the other hand, in ONF's SDN architecture [Ope14b] framework, network resource orchestration can be recursively programmed by the means of virtualizers and agents through the I-CPI interface (see Fig. 23). Virtualizers are responsible for both i) creating client specific virtual topologies and resources and for ii) client specific policy enforcements.

We argued through the introduction of the UNIFY services offered at the Cf-Or reference that a unified compute and network programmatic interface together with policy enforcement offered directly to the VNFs could enable VNF agnostic, automated elasticity control in a loop closest to the resources.

We will use the IDS example introduced in Fig. 14 to discuss different approaches to realize service elasticity.

6.3.1 Deployment scenarios

Let's have a look at the different deployment scenarios depicted in Fig. 25 and 26. As a general note, signaling messages are labeled and shown by arrow-heads in the figures, but we assume that there is always an underlying logical network connection. Also, the arrowheads only illustrate our discussion of the primarily control message flow, but it is assumed that control messages also pass in the reverse direction. Even though the two figures differ in their components we describe the initial service deployment by concurrently denoting the corresponding components like ETSI/UNIFY. Fig. 25 shows an example ETSI MANO set-up and Fig. 26 shows an example UNIFY set-up.

First of all, in the upper part of Fig. 25 and 26 we show the user's view of the example network service: the cascade of an Elastic IDS and a Broadband Network Gateway (BNG) connected to the user's SAP and to the Internet. According to the service definition and decomposition model introduced in Fig. 14 the Elastic IDS needs connection to the respective

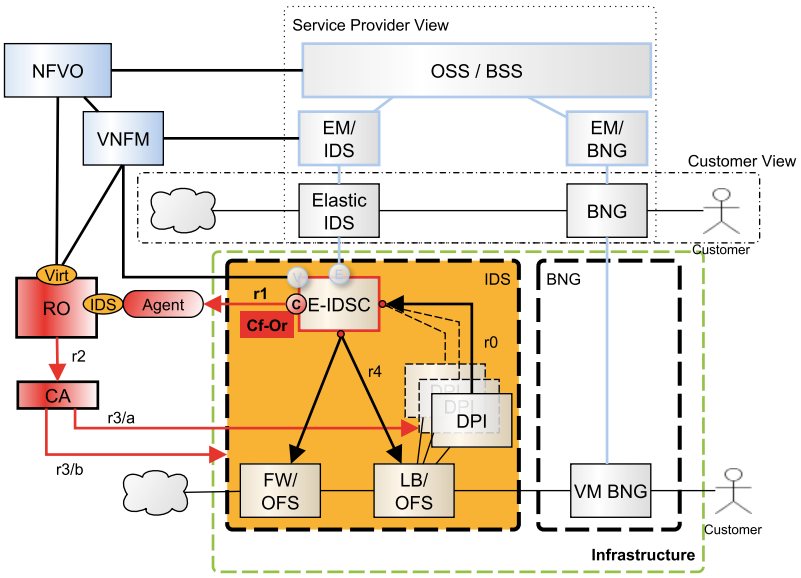

Figure 26: Elastic control loop according to the UNIFY framework

EM and OSS. Let us assume that the BNG is also connected to an EM and an OSS. Let's assume that the network service request is issued through the BSS, translated to the OSS, who requests the allocation of the network service from the NFVO.

In the ETSI case, any service decomposition must happen in the NFVO and a fully specified VNF-FG with associated resource mapping must be sent to the VIM. The VIM allocates the VNF resources and the forwarding overlay resulting in Fig. 25.

In the UNIFY case, thanks to the logically centralized RO, an abstract network service graph can be sent to the RO by the means of an NF-FG. The RO, considering the VNF decomposition model as presented in Fig. 14, can decide which option to initiate based on the available infrastructure resources. We must emphasis, that the services offered by the different options are equivalent; the choice in the RO can be based on operational policies like energy efficiency, utilization, etc.

Additionally, the RO must detect if any VNF definition contains a C interface, which must be resolved to a connection and a virtualization at the RO's Cf-Or reference point. For example, in the case of Fig. 26, the RO must automatically create a virtualized domain view equivalent to the orange IDS box (note that BNG components are excluded) and connect the E-IDSC's C interface to the corresponding Cf-Or agent. Note also, that the number of external interfaces related to a service component does not change during the decomposition process (see interfaces 1 and 2 in Fig. 14). The C interface of a VNF is internally resolved to a connection to the corresponding RO. If there are multiple levels of virtualization domains, then the interface C is resolved when first seen by an RO. The connection between such C interfaces and the corresponding RO is included into the NF-FG description of the service. Therefore, a proper network overlay has to be allocated to this control network as part of the NF-FG orchestration. Finally, once the RO is done with the full mapping of the NF-FG to the infrastructure resources, than it instructs the CA to instantiate VNFs and the network overlay.

Last but not least, in both cases, EMs and the OSS must handle any remaining service configurations.

6.3.2 The ETSI setup

According to the ETSI NFV view a VNFM or a NFVO is in charge of the scaling of the service. Let's assume that the E-IDSC learns that DPIs cannot sustain the user traffic ($v0$). E-IDSC notifies the VIM through VeNf-Vnfm ($v1$) about the need of extra resources. Since VNFM deals only with VNFs and is service logic agnostic, it cannot know what further components to deploy, but must let the NFVO know ($v2$) about the problem. Note here, that even the scale up/down of a single VM looks problematic, as the resources to be increased belong to the DPI and not to the E-IDSC. Assume, that the NFVO has to perform a scale out. The NFVO must look-up the service specific template to get an idea how to add more resources. Once it learns that a new DPI should be instantiated and connected into the service, it can allocate resources and send the request ($v3$) to the VIM. The VIM can execute the allocation by instantiating ($v4/a$ and $v5/a$) another DPI and re-configuring ($v4/b$ and $v5/b$) the forwarding overlay.

It is important to note that the E-IDSC (VNF in general) must learn that the scaling operation is done, e.g., to be able to update the LB configuration ($v6$). Therefore, there must be a reverse flow of control information from the VIM, through (NFVO), VNFM, (EM) to the control application.

6.3.3 The UNIFY setup

According to the UNIFY architecture, we can connect interface C of the E-IDSC to the RO's realizing the Cf-Or reference point (see Fig. 26). Let's assume again, that the E-IDSC learns that DPI is running out of resources ($r0$). The E-IDSC knows the service logic as it is designed and developed as part of it. It knows its instantiated VNFs, the corresponding forwarding overlay and allocated resources by the means of the NF-FG abstraction conveyed through the Cf-Or reference point. Therefore, the E-IDSC is in the best position to initiate scale up/down or scale out/in. By the means of the virtualizer and the agent at the RO, the E-IDSC can issue an update request ($r1$) to its sub-set of the deployed NF-FG.

Once the RO receives the NF-FG update request ($r1$) it has to verify the associated client policies and then map the change request to the available resources. Then, the RO sends ($r2$) the joint compute and network mapping to CA, who executes it through ($r3/a$) and ($r3/b$) for the DPI instantiation and for the overlay update respectively. Then E-IDSC can update the LB configuration ($r4$).

6.3.4 Discussions

Regarding both set-ups all the user plane, the control plane and management plane network overlays must be created dynamically. For example, the DPI shall not only be connected to the user plane traffic overlay, but it must also be connected to the E-IDSC's proper interface.

In the case of ETSI operation, the NFVO knows the updated VNF graph, however, how the E-IDSC learns the revised structure is unspecified. In the UNIFY case, the revised NF-FG is issues by the E-IDSC itself, so upon successful completion, it will know the structure in the first hand.

Another issue is who should configure the LB to split the load among the DPIs? If such scaling is non-transparent to the involved network functions (e.g., stateful processing), then a service logic must assist the operational re-configuration of components. The question is, can an external virtualization management component (e.g., NFVO, VNFM or RO) bear with such a logic? We believe that such logic is best placed into the control application. This is similar to the end-to-end principle of Internet, which says that application specific functions should reside in the application (control application) rather than within the network (virtualization managers).

Regarding security issues, in the ETSI set-up the signaling goes through trusted elements like EM, OSS, VNFM, NFVO and VIM. In this case, it is relatively easy to enforce user policies related to the service contract due to the involvement of service layer management functions. In the case of the UNIFY approach, however, resource policies related to the service contract must be enforced in the corresponding virtualizer, which operates at compute and storage abstractions and has no notion of the user service logic. For example, in the virtualizers associated to the E-IDSC, the RO could limit

- the available resources (max 10 virtual CPU, max 5 Gbyte RAM, less than 100Mbps in/out rate, maximum cost of resources, etc.),
- the capabilities as available VNF types (DPI and OFS) for the control application.

Such policy information could be extracted from the genuine service request and/or from client policies (e.g., maximum cost, etc.). The main issue here is that unlike the NFVO, VNFM, EM and OSS, the control application is not a trusted infrastructure component.

We believe that the direct elastic control interface (Cf-Or) of UNIFY will bring values $i)$ to the users by the possibility of new and innovative elastic services; $ii)$ to the developers by better control of their service behavior and also $iii)$ to the resource owners by better serving the resource needs of their consumers.

7 Summary

In this deliverable we summarized and concldued the UNIFY architecture in three details corresponding to an overarching view, functional view and implementation based system designs. While the document is titled "final architecture" and it already contains learnings while doing proof of conecept prototypes, we would like to reserve the right to revise the presented archicture based on our later learnings based on implementations.

Since this document focuses solely on the architecture description, we have to point the reader to [D2.1] for state of the art reviews, description of use cases, definition of requirements and design principles.

Our main contributions are

- the split of the architecture into service, orchestration and infrastructure layers;
- the definition of reference points between functional components;
- the definition of the Sl-Or reference point for joint virtualization and orchestration of software and networking resource
- allowing multi-level hiearchies of UNIFY domains;
- the definition of the direct Cf-Or elastic resource control interface for NFs;
- an approach to combine the static virtualization views with dynamic resource monitoring.

The introduction of network and service functionality virtualization in carrier-grade networks promises improved operations in terms of flexibility, efficiency, and manageability. In current practice, virtualization is controlled through orchestrator entities that expose programmable interfaces according to the underlying resource types. Typically this means the adoption of established data center compute/storage and network control APIs, which were originally developed in isolation. Arguably, the possibility for innovation highly depends on the capabilities and openness of the aforementioned interfaces.

We introduced an architecture which combines compute, storage and network resources into a joint programmatic reference point, allowing multi-level virtualization and orchestration of Network Function Forwarding Graph for fast and flexible service chaining. The UNIFY architecture and the UN pave the way for low cost generic purpose equipment anywhere in the network by leveraging all types of compute, storage and networking resources they offer.

We defined a SP-DevOps framework to complement resource virtualization and orchestration for both NF and service developers with monitoring orchestration, NF-FG troubleshooting and verification.

References

[5GP13] 5G-PPP Association. Contractual Arrangement: Setting up a Public-Private Partnership in the Area of Advance 5G Network Infrastructure for the Future Internet between the European Union and the 5G Infrastructure Association. Dec. 17, 2013. URL: http://5g-ppp.eu/contract/.

[Cso+14a] Attila Csoma, Balázs Sonkoly, Levente Csikor, Felicián Németh, András Gulyás, Dávid Jocha, János Elek, Wouter Tavernier, and Sahel Sahhaf. "Multi-layered service orchestration in a multi-domain network environment. Demonstation." In: 2014 Third European Workshop on Software-Defined Networks (EWSDN). http://ewsdn.eu, 2014.

[Cso+14b] Attila Csoma, Balázs Sonkoly, Levente Csikor, Felicián Németh, András Gulyás, Wouter Tavernier, and Sahel Sahhaf. "ESCAPE: Extensible Service ChAin Prototyping Environment using Mininet, Click, NETCONF and POX. Demonstation." In: Proceedings of the 2014 ACM conference on SIGCOMM. ACM. 2014, pp. 125–126.

[D2.1] Unify project. Deliverable 2.1: Use Cases and Initial Architecture. Tech. rep. UNIFY Project, 2014.

[D3.1] Wouter Tavernier et al. D3.1 Programmability framework. Tech. rep. D3.1. UNIFY Project, Oct. 2014.

[D4.1] Pontus Sköldström et al. D4.1 Initial requirements for the SP-DevOps concept, Universal Node capabilities and proposed tools. Tech. rep. D4.1. UNIFY Project, Aug. 28, 2014.

[D4.2] Catalin Meirosu et al. Deliverable 4.2: Proposal for SP-DevOps network capabilities and tools. Tech. rep. Work in Progress. UNIFY Project, 2015. URL: to%20be%20announced.

[D5.1] Hagen Woesner et al. Deliverable 5.1: Universal Node functional specification and use case requirements on data plane. Tech. rep. UNIFY Project, 2014. URL: https://www.fp7-unify.eu/files/fp7-unify-eu-docs/Results/Deliverables/UNIFY-WP5-D5.1-Universal%20node%20functional%20specification.pdf.

[D5.2] Hagen Woesner et al. D5.2 Universal Node Interfaces and Software Architecture. Tech. rep. UNIFY Project, Aug. 2014. URL: http://fp7-unify.eu/files/fp7-unify-eu-docs/Results/Deliverables/UNIFY-WP5-D5.2-Universal%20node%20interfaces%20and%20software%20architecture.pdf.

[Enn+11] R. Enns, M. Bjorklund, J. Schoenwaelder, and A. Bierman. Network Configuration Protocol (NETCONF). RFC 6241 (Proposed Standard). Internet Engineering Task Force, June 2011. URL: http://www.ietf.org/rfc/rfc6241.txt.

[ETS13a] ETSI. Architectural Framework. English. Group Specification v1.1.1. ETSI, Oct. 2013, pp. 1–21. URL: http://docbox.etsi.org/ISG/NFV/Open/Published/gs_NFV002v010101p%20-%20Architectural%20Fwk.pdf.

[ETS13b] ETSI. White Paper: Network Functions Virtualisation (NFV). 2013. URL: http://portal.etsi.org/NFV/NFV%5C_White%5C_Paper2.pdf.

[ETS14] ETSI. Network Function Virtualization (NFV) Management and Orchestration. Group Specification V0.6.1. (DRAFT). ETSI, July 2014, pp. 1–196. URL: http://docbox.etsi.org/ISG/NFV/Open/Latest%5C_Drafts/NFV-MAN001v061-%20management%20and%20orchestration.pdf.

[HZH14] Changcheng Huang, Jiafeng Zhu, and Peng He. SFC Use Cases on Recursive Service Function Chaining. Internet-Draft draft-huang-sfc-use-case-recursive-service-00.txt. IETF Secretariat, July 2, 2014.

[Joh+13] W. John et al. "Research Directions in Network Service Chaining". In: Future Networks and Services (SDN4FNS), 2013 IEEE SDN for. 2013. DOI: 10.1109/SDN4FNS.2013.6702549.

[Koh+00] Eddie Kohler, Robert Morris, Benjie Chen, John Jannotti, and M Frans Kaashoek. "The Click modular router". In: ACM Transactions on Computer Systems (TOCS) 18.3 (2000), pp. 263–297.

[LHM] B. Lantz, B. Heller, and N. McKeown. "A Network in a Laptop: Rapid Prototyping for Software-defined Networks". In: ACM HotNets 2010. URL: http://mininet.org/.

[Mei+14] Catalin Meirosu, Antonio Manzalini, Juhoon Kim, Rebecca Steinert, Sachin Sharma, and Guido Marchetto. DevOps for Software-Defined Telecom Infrastructures. Internet-Draft draft-unify-nfvrg-devops-00.txt. IETF Secretariat, Oct. 27, 2014.

[ONF14] ONF. Open Networking Foundation. 2014. URL: https://www.opennetworking.org/.

[Ope13] Open Networking Foundation. OpenFlow Switch Specification 1.4.0. Oct. 2013. URL: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.4.0.pdf.

[Ope14a] Open Networking Forum (ONF). OpenFlow Specifications. web. 2014. URL: https://www.opennetworking.org/sdn-resources/onf-specifications/openflow.

[Ope14b] Open Networking Forum (ONF). SDN architecture. TR (Technical Reference), non-normative, type 2 SDN ARCH 1.0 06062014. ONF, June 2014. URL: https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/TR_SDN_ARCH_1.0_06062014.pdf.

[Ope14c] OpenStack. OpenStack Compute API v2 Reference. Tech. rep. OpenStack, Oct. 2014. URL: http://developer.openstack.org/api-ref-guides/bk-api-ref-compute-v2.pdf.

[POX] The POX Controller. 2014. URL: https://github.com/noxrepo/pox.

[Ris+14] Fulvio Risso, Ivano Cerrato, Alex Palesandro, Tobias Jungel, Marc Sune, and Hagen Woesner. "Demo: User-specific Network Service Functions in an SDN-enabled Network Node". In: 2014 Third European Workshop on Software-Defined Networks (EWSDN). http://ewsdn.eu, Sept. 2014.

[TM 12] TM Forum. SLA Management Handbook. English. Tech. rep. BG917, Release 3.1, Approved Version 1.2. TM-Forum, Nov. 2012, pp. 1–127. URL: http://www.tmforum.org/GuideBooks/GB917SLAManagement/48401/article.html.

[Vol+00] J. Vollbrecht, P. Calhoun, S. Farrell, L. Gommans, G. Gross, B. de Bruijn, C. de Laat, M. Holdrege, and D. Spence. AAA Authorization Framework. RFC 2904 (Informational). Internet Engineering Task Force, Aug. 2000. URL: http://www.ietf.org/rfc/rfc2904.txt.

[XACML] eXtensible Access Control Markup Language (XACML) Version 3.0. 2013. URL: http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.pdf.