5GCity

Grant Agreement No.761508 5GCITY/H2020-ICT-2016-2017/H2020-ICT-2016-2

D4.1: Orchestrator design, service programming and machine learning models

Dissemination Level		
☒	PU:	Public
☐	PP:	Restricted to other programme participants (including the Commission Services)
☐	RE:	Restricted to a group specified by the consortium (including the Commission
☐	CO:	Confidential, only for members of the consortium (including the Commission Services)

Grant Agreement no: **761508**	Project Acronym: **5GCity**	Project title: **5GCity**

Lead Beneficiary: I2CAT Foundation	Document version: **V1.0**

WP4 – Scalable Management & Orchestration, and Service Programming Models

D4.1: Orchestrator design, service programming and machine learning models

Start date of the project: 01/06/2017 (duration 30 months)	Contractual delivery date: M12	Actual delivery date: Date of submission to Coordinator

Editor name: Hamzeh Khalili and Apostolos Papageorgiou (i2cat)

List of Contributors

Participant	Short Name	Contributor
I2CAT Foundation	I2CAT	Hamzeh Khalili, Apostolos Papageorgiou, Shuaib Siddiqui, Julio Barrera
NEC	NEC	Felipe Huici, Kenichi Yasukata
NEXTWORKS	NXW	Nicola Ciulli, Paolo Cruschelli, Elian Kraja, Elio Francesconi
UBIWHERE LDA	Ubiwhere	Ricardo Preto
ITALTEL SPA	ITL	Antonino Albanese, Viscardo Costa
University of Bristol	UINIVBRIS	Carlos Colman Meixner
ADLINK TECHNOLOGY SARL	PRISMTECH	Gabrielle Baldoni
Virtual Open Systems	VOSYS	Teodora Sechkova, Michele Paolino

List of Reviewers

Participant	Short Name	Contributor
Virtual Open Systems	VOSYS	Michele Paolino
University of Bristol	UINIVBRIS	Carlos Colman Meixner
I2CAT Foundation	I2CAT	Sergi Figuerola
NEC	NEC	Felipe Huici

Change History

Version	Date	Partners	Description/Comments
0.1	05-03-2018	I2CAT	First draft by i2CAT
0.2	11-04-2018	I2CAT	Second draft, including consolidated ToC and initial contents
0.3	27-04-2018	I2CAT	Added main bodies of sections 2, 3, and 4 (Orchestrator, SDK, and FML)
0.4	07-05-2018	NXW	Corrections in section 3 (SDK)
0.5	10-05-2018	PRISMTECH	Added descriptions about MEC components
0.6	11-05-2018	NXW	Updates of interface documentation, information model, and more in section 3 (SDK)
0.7	14-05-2018	I2CAT	Added introduction and conclusion, various updates in all technical sections
0.8	18-05-2018	I2CAT	Addressed internal reviews, updates interfaces and figures, consolidated inputs, and more
0.9	24-05-2018	I2CAT	Cross-checked sections, fine-tuned diagrams, consolidated references, and more
0.10	25-05-2018	I2CAT	Version for project and technical coordinators' check
0.11	30-05-2018	I2CAT	Addressed coordinators' reviews and fine-tuned architecture figures and terminology
1.0	31-05-2018	I2CAT	Clean version for submission

DISCLAIMER OF WARRANTIES

This document has been prepared by 5GCITY project partners as an account of work carried out within the framework of the contract no 761508.

Neither Project Coordinator, nor any signatory party of 5GCITY Project Consortium Agreement, nor any person acting on behalf of any of them:

- makes any warranty or representation whatsoever, express or implied,
  - with respect to the use of any information, apparatus, method, process, or similar item disclosed in this document, including merchantability and fitness for a particular purpose, or
  - that such use does not infringe on or interfere with privately owned rights, including any party's intellectual property, or
- that this document is suitable to any particular user's circumstance; or
- assumes responsibility for any damages or other liability whatsoever (including any consequential damages, even if Project Coordinator or any representative of a signatory party of the 5GCITY Project Consortium Agreement, has been advised of the possibility of such damages) resulting from your selection or use of this document or any information, apparatus, method, process, or similar item disclosed in this document.

5GCITY has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 761508. The content of this deliverable does not reflect the official opinion of the European Union. Responsibility for the information and views expressed in the deliverable lies entirely with the author(s).

Table of Contents

Figures

Tables

Executive Summary

This document describes the **components of the 5GCity architecture related to orchestration, service programming, and machine learning** as main outcomes of tasks T4.1, T4.2, and T4.3. The overall 5GCity architecture is described in Deliverable D2.2 [1] and based on pilot requirements introduced in Deliverable D2.1 [2]. Our orchestration, service programming, and machine learning components are vital for addressing challenges of state-of-the-art 5G orchestrators and platforms, such as multi-tenancy support and efficient configuration and resource placement. In the main body of this document:

Firstly, we describe the 5GCity platform design, including a scalable orchestration solution that is based on the ETSI NFV MANO specification, extending it to support network slicing, intelligent SLA (Service-Level Agreement) management, efficient resource allocation, a broader palette of underlying infrastructure technologies, and more. This 5GCity orchestrator is the core management component of the functional 5GCity architecture. In this context, we also discuss about the design of the 5GCity dashboard, which is an entry point to the 5GCity platform in a way that provide capabilities to deploy end-to-end services upon a 5G infrastructure that follows the “neutral host” scenario.

Secondly, we provide a new SDK (Service Development Kit) and service programming model following the flow-based programming paradigm to enable rapid edge service programming directly on the distributed city edge infrastructure. This SDK can be used by “customers” of the neutral host (i.e., tenants such as Network Operators and Content/Service Providers) to request slices with combine edge computing resources, 5G access networking resources, and more.

Thirdly, machine learning- and optimization-based location-aware algorithms are being developed and documented, with the intention to achieve VNF (Virtual Network Function) placement, re-deployment, and re-configuration times that stay within the requirements of the neutral host scenario and the project Use Cases within an edge-based city environment.

Therefore, the modules involved in these three parts are detailed in this deliverable, including their components and internal architecture, functionality, external and internal interfaces, interaction with other components, and the main high-level workflows. Finally, we conclude with a summary of how our developments advance the state of the art of 5G orchestration, service programming, and machine learning for edge computing.

1. Introduction

The main goal of 5GCity is to define how to turn a city into a distributed and multi-tenant edge infrastructure which uses orchestration and service programming to integrate 5G services administered by a neutral host. These 5G services will be used by municipalities to host Smart City services, by Virtual Network Operators (VNO) to extend their networks, and by additional third party providers such as media or automotive verticals to offer innovative services to their customers.

Before we introduce our proposed components for 5G- and "neutral host"-driven orchestration, service programming, and edge machine-learning, we start by briefly summarizing the 5GCity architecture (focussing on related components) and introducing the main (orchestration and service programming) challenges that we are trying to solve.

1.1. 5GCity architecture summary

The 5GCity architecture contains functional blocks that combine distributed cloud technologies and edge network virtualization for exploiting the new city edge infrastructure deployments such as street cabinets, city-owned edge servers, wireless access points, small cells, and more. Figure 1 summarizes our proposed four layers, which 5GCity architecture introduced in [1]. The orchestration, control, and service programming functionalities, which are the focus of this Deliverable, are mainly implemented by the 5GCity Platform, but also the SDK and the VIMs.

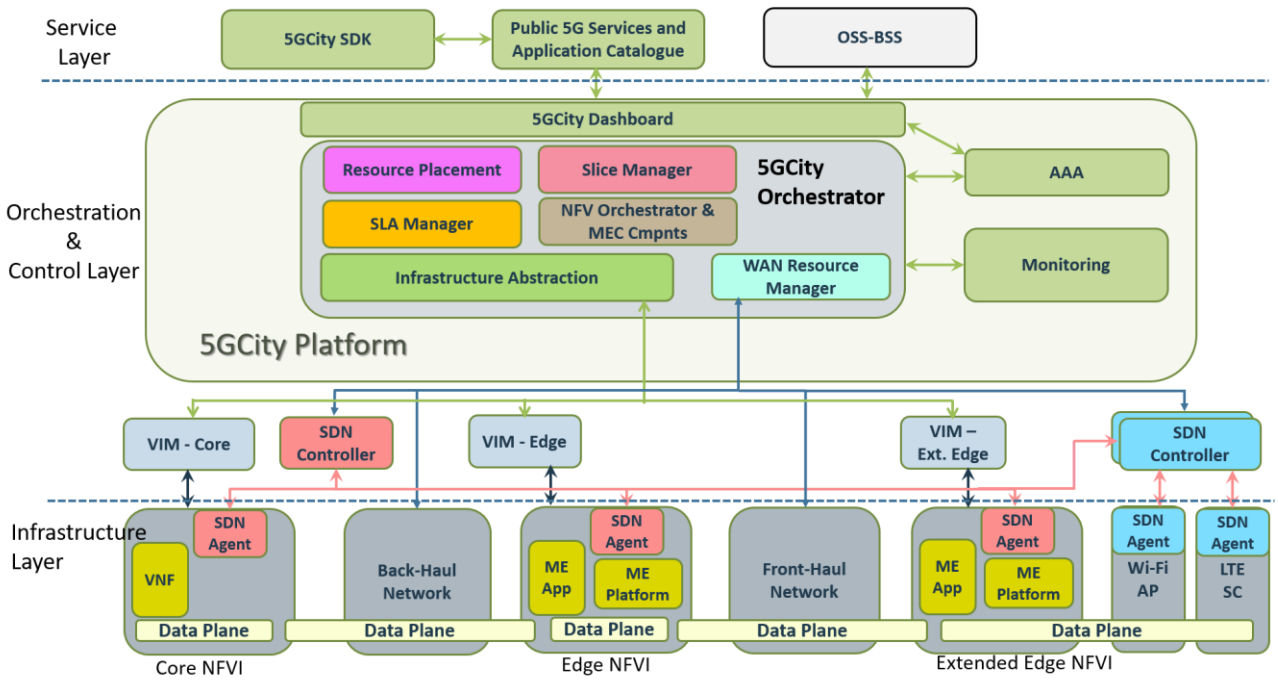

Figure 1. 5GCity high-level architecture design

Service layer consists of a set of functions/tools available for infrastructure providers, customers, and any third party. The sets of tools can be identified as:

- **SDK** and service programming tools create, validate, and test specific VNFs, and network services packages to be deployed by dashboard users upon 5GCity physical infrastructure.

- Public **catalogue** of 5G services and applications, as well as an **OSS/BSS system** (Operation/Billing Support System), which should be implemented by reusing existing technologies.

Orchestrator and Control layer is the core management layer in the 5GCity architecture. This layer performs the lifecycle management and orchestration of all 5G-based edge services and controls the available city edge infrastructure. It includes the following main functional parts:

- **The Dashboard** represents the entry point to the system, used by administrators of the cloud-like 5G infrastructure to connect to the orchestrator functionality through northbound API (Application Programming Interface). It provides capability to deploy end-to-end services upon the city edge infrastructure. The dashboard will hold different views to enable multi-tenancy by allowing different roles and capabilities for each user or tenant (e.g. super user, tenant administrator, external user).
- **The Orchestrator** represents the core functionality of the architecture which provides the capability to 1) manage a non-homogeneous set of physical resources (i.e. computing, storage, wired network and wireless network), 2) abstract physical resources 3) operate a horizontal slicing thus providing inherent and 4) cast end-to end services tailored to a multitenant framework.
- **The Monitoring component** collects and processes runtime performance and status data from resources and services.
- The **Authentication, Authorization, and Accounting (AAA)** component performs the required security-related tasks.

Between the orchestration and the Infrastructure layer reside the elements that virtualize and manage cloud and radio access technologies in the distributed city edge infrastructure. The joint deployment of the following elements supports the creation an open, multi-tenant virtualization platform running on heterogeneous devices as well as virtualizing the underlying edge network (both wired and wireless):

- **VIMs (Virtualized Infrastructure Manager)** control and manage the NFVI (Network Function Virtualization Infrastructure) compute, storage and network resources, usually within an operator's infrastructure domain. A VIM can be specialized in handling a certain type of NFVI resource (e.g. compute-only, storage-only, networking-only), or can be capable of handling multiple types of NFVI resources (e.g. in NFVI-Nodes). In the 5GCity architecture three different types of VIM will be used:
    - VIM-Core operates homogeneous physical resources located in the city data center.
    - VIM-Edge operates a non–homogenous, wide area, resource-constrained set of physical resources located in street cabinets across the City.
    - VIM-Extended Edge operates a non-homogeneous and resource-constrained set of devices located at the extended edge (e.g. lamppost), as well as any IoT (Internet of Things) sensors.
- **SDN Controllers** provide intuitive programmatic interfaces along the lines of the network interfaces and thus can be best categorized as an abstraction layer below the VIM for a given NFVI-PoP (Point of Presence). Each NFVI-PoP or administrative domain may include a network controller, e.g. SDN (Software-Defined Networking) controller, responsible for providing the programmable network interfaces that enable the establishment of connectivity within the domain. In the 5GCity architecture, the SDN controllers are defined as a master and slaves model.

The Infrastructure layer is the layer for physical resources management across the city edge infrastructure.

- **Core NFVI (Data Center)** manages the virtualized and non-virtualized resources, supporting full and partially virtualized network functions. Virtualized resources in-scope are those that can be associated with virtualization containers, and have been catalogued and offered for consumption through abstracted services, for example:
    - Compute, including machines (e.g. hosts or bare metal), and virtual machines, as resources that comprise both CPU (Computer Processing Unit) and memory.

- Storage, including volumes of storage at either block or file-system level.
- Network, including: networks, subnets, ports, addresses, links and forwarding rules, for the purpose of ensuring intra- and inter-VNF connectivity.

- **Back-haul network** is a portion of network that provides connectivity from MEC (Multi-access Edge Computing) nodes to the DC (Data Center) resources.
- **Edge NFVI** (MEC nodes) are non-homogeneous pools of resources located away from centralized DCs, with limited capabilities due to constrained resources in terms of power, computing, connectivity.
- **Front-haul** network is a portion of network that connects the cell site unit (Remote Radio Head/RRH or Radio Unit/RU) of the base station to its digital unit (BBU) residing centrally in a DC or local cabinet.
- **Extended Edge NFVI** (including Small Cells/Wi-Fi) corresponds to the hardware equipment located at the extended edge, e.g., at the lamppost, which is also used to produce the LTE/5G radio channels. 5GCity, following the evolution of radio access network towards a perfect integration with 5G topics, foresees a Wi-Fi and LTE/5G as main radio access technologies.

1.2. Challenges in 5G Orchestration and Service Programming

Orchestration and service programming (in the context of networking) refer to the on-boarding, instantiation, and lifecycle management of network functions. This work is mainly concerned with virtualized network functions. As such, orchestrators (or orchestrator platforms) are the management components that perform the aforementioned tasks, usually empowered by consistent and complete data models of the involved entities, e.g., Network Services (NS) and Virtual Network Functions (VNF).

State-of-the-art orchestration and service programming solutions (which are analysed in Appendix A) have been developed in scenarios in which:

i. Network slices are created and instantiated with very low frequency and loose time requirements

ii. The management of computing and storage resources of the slices is decoupled by the management of networking resources

iii. The services and VNFs running on the slices are placed and configured based on algorithms that focus on optimality with regard to global runtime metrics such as latency times and bandwidth consumption.

Although the above are valid assumptions in many scenarios, the neutral host scenario and the other 5GCity Use Cases described in D2.1 introduce a higher dynamicity in slice lifecycle management, a higher diversity of slice elements (Small Cells, MEC nodes, WiFi APs, as well as DC nodes, networking equipment etc.), and tighter timing requirements for slice and service (re-)instantiation and (re-)allocation. This forces us to face the following main challenges:

- CHALLENGE 1: The “slicing-unaware” and “Cloud-oriented” design of state-of-the-art NFV orchestrators restricts the efficiency of the management of neutral host scenarios by posing restrictions with regard to i) the number and the diversity of NFVI technologies that can be managed, ii) the degree of isolation of the orchestrated Network Services and VNFs of different stakeholders, iii) optimality of resource allocation and fragmentation, and iv) the diversity of captured and analysed runtime VNF parameters.
- CHALLENGE 2: For Service Programming, Service Development Kits (SDK) are a fundamental component to open-up the virtualization advantages. An SDK is a stand-alone collection of services, functionally integrated with an orchestration platform, able to craft network service templates ready to be deployed over a pool of virtual resources. Different SDK toolkits are already available within the NFV (Network Function Virtualization) realm (e.g. the COHERENT SDK, the SONATA SDK, etc.).

However, most of these SDKs adopt a network-centric approach, aiming at defining and testing Network Services and VNFs before their instantiation in runtime MANO (Management and Orchestration) environments. The user of those toolkits needs to be fully aware of the complex details of the information models used within the orchestration platform. Also those state-of-art toolkits are built on top of specific frameworks and bound to be used in those specific deployments, thus missing the capability to encompass several NFV technologies.

- CHALLENGE 3: The state of the art in computing (whether learning or prediction) NN (Neural Networks) is to assume that the infrastructure is entirely homogeneous, consisting of clusters of equally-spec'd servers with well-provisioned network links between them. Some of the previous challenges will be hard to handle without advanced intelligence, so the usage of AI might be essential. However, the right tools and learning strategies are not obvious.

After the description of the developed technologies, we will come back to these challenges, summarizing how they are being addressed by our components and mechanisms.

2. 5GCity Orchestration and Control

This section introduces our proposed 5GCity orchestration and control platform design, development, and implementation. Our platform will be capable to operate large number of devices in dynamic conditions (e.g. network, device overload, multi-tenancy etc.) in an efficient manner. The 5GCity orchestration platform is a core management component of the functional 5GCity architecture. It is responsible for the lifecycle management and orchestration of all 5G-based edge services and for the control and management of the available city edge infrastructure. It also includes 5G edge service programming models, as well as a northbound API to enable access to the different edge services and the orchestrator functionalities.

2.1. Orchestration

2.1.1. Existing orchestration platforms

An investigation of existing ETSI (European Telecommunications Standards Institute) NFV-based orchestration platforms has been performed in order to see if there is a possibility to build upon existing functional modules and integrate edge computing and 5G networking aspects with traditional orchestration workflows. Among others, we have analysed platforms like Tracker [3] TeNoR [4], OpenBaton [5], Cloudify [6], ONAP [7], SONATA [8], and OSM (OpenSourceMano – [9]). The detailed discussion of these platforms, together with views of their functional architectures, are provided in Appendix A.

The 5GCity orchestration platform will leverage on existing orchestration tools to guarantee performance and scalability in dense urban 5G deployments cities. Therefore, based on the analysis presented in Appendix A, the project has chosen to utilize and extend the ETSI OSM for orchestration because of the following reasons:

1. Fine-grained architecture.
2. Open source platform with standardized interfaces, which allows third parties to extend functions/plugins and/or add new ones.
3. Supports the addition of other types of VIM, SDN controller and monitoring components.
4. Can be extended for multi-VIM and multi-cloud environments such as OpenStack (almost all the versions), OpenVIM, VMware, vCloud, Fog05, etc.
5. Can integrate multiple SDN controllers such as OpenDayLight, ONOS, Floodlight, etc.
6. Widely adopted in 5G research projects.
7. Regular software updates, i.e., every 6 months.
8. High availability of software and code documentation and distributions.

2.1.2. Architecture

This section describes the architecture of the orchestration platform proposed in the 5GCity project. 5GCity orchestrator shall comply with ETSI NFV MANO specifications, while adapting to specific distributed edge infrastructures and 5G technologies, including network slicing. The orchestrator platform provides a unified management of connectivity with end-to-end services for dynamic provisioning of network slices for accommodating smooth, automated introduction, and deployment of novel services. This allows network operators to seamlessly control and orchestrate services for different vertical, which are provided over multiple platforms.

Before presenting the orchestration architecture designed for 5GCity, we present a view of 5G hardware resources that are important for the 5GCity project upon an ETSI NFV orchestration landscape. This view was based on a study of 3GPP (3d Generation Partnership Project) from December 2015 about introducing ETSI NFV MANO elements in to the 3GPP architecture (3GPP Rel. 13 Study: 32.842 recommendation). The idea was to come up with the mixed network management mapping relationship between 3GPP and the ETSI NFV MANO architecture framework. Figure 2 presents this view and served as a basis for the design principles of the 5GCity orchestrator.

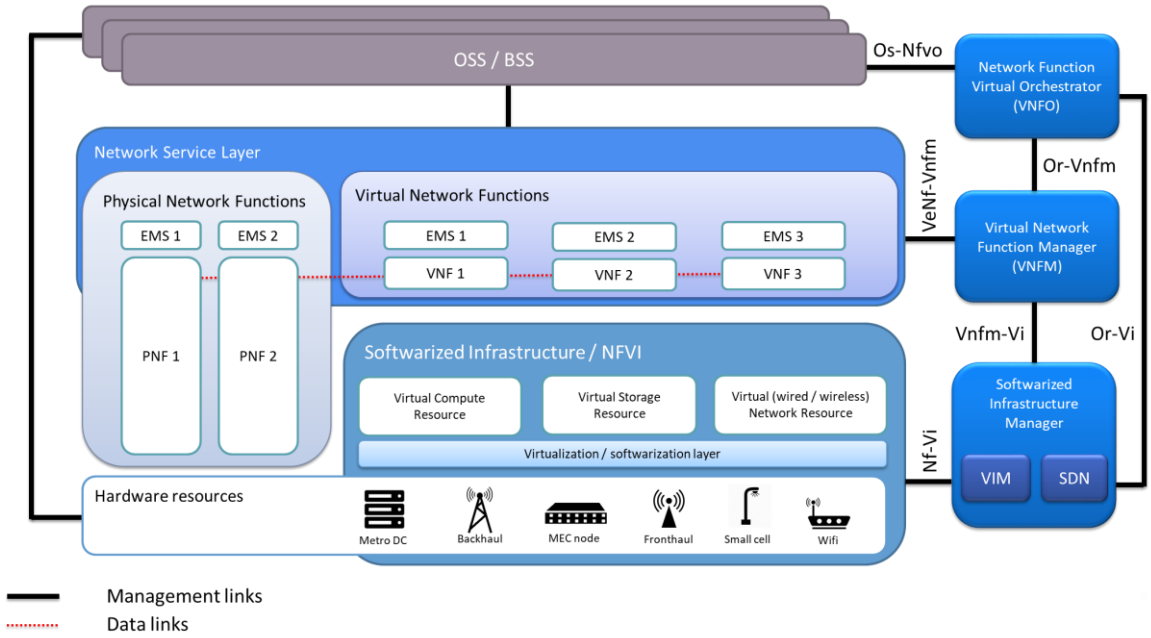

Figure 2. A view of 5G hardware resources upon the ETSI NFV orchestration landscape

Figure 3 introduces the 5GCity orchestrator in accordance with the ETSI OSM model. The extra functionalities compared to OSM will be mainly reflected in the internal logic and the interactions of the following main components of the 5GCity Orchestrator (which are detailed in the next subsection):

- NFV Orchestrator
- MEC Components
- Resource Placement
- SLA Manager
- Slice Manager
- Infrastructure Abstraction
- WAN Resource Manager

The Dashboard and the Monitoring tightly interact with the 5GCity Orchestrator and are very important for the orchestration procedures, but they do not belong to the architecture or the scope of the orchestrator, thus they are discussed in separate sections.

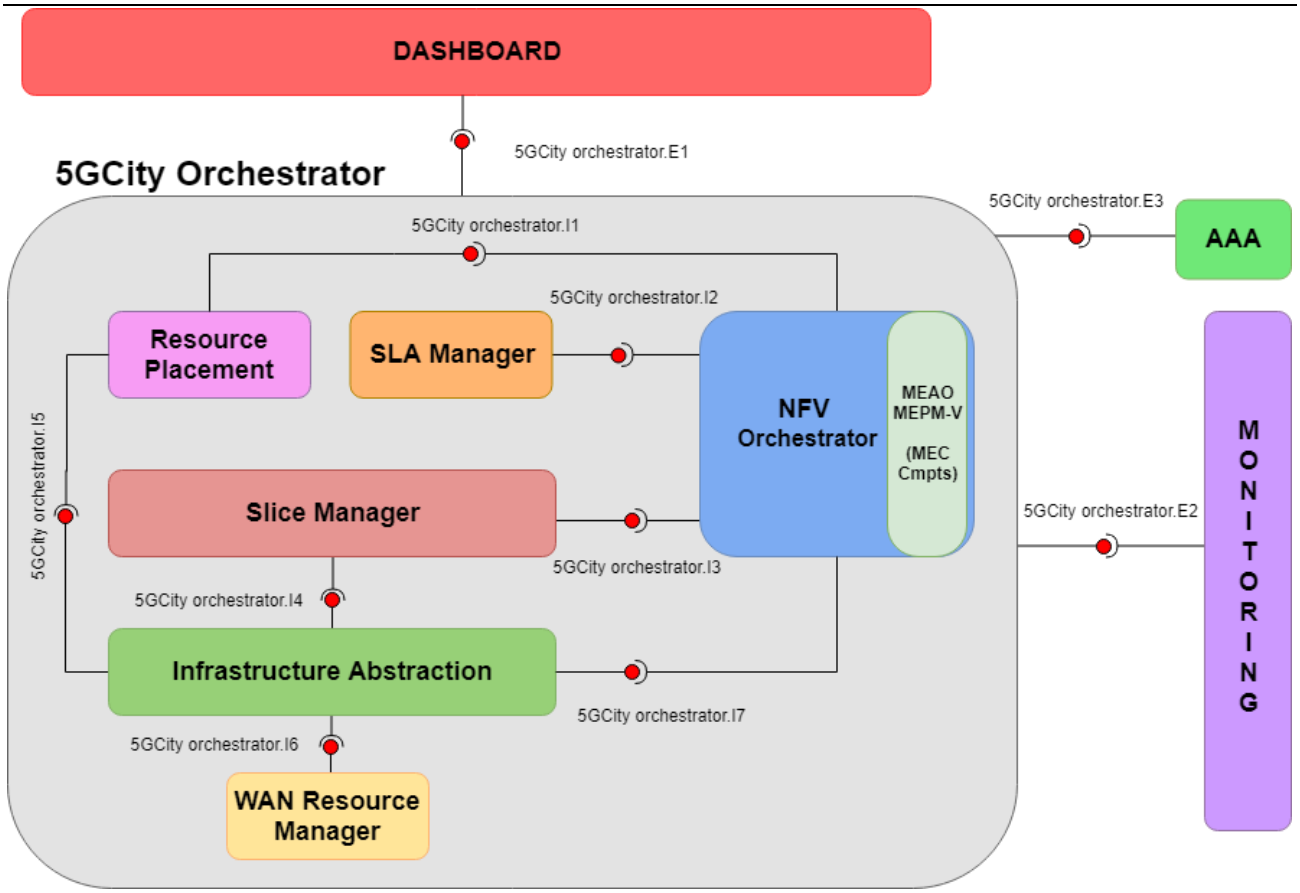

Figure 3. 5GCity Orchestration platform architecture

2.1.3. Components

NFV Orchestrator (NFVO)

As already mentioned, for the NFVO (NFV Orchestrator) functionality the project will adopt the OSM solution, extending it to support (and/or incorporate) all the other components and functionalities described in the following paragraphs. In its core, the NFVO is responsible for NS on-boarding and VNF management, as described in the respective ETSI NFV specifications.

MEC components

Due to the edge-computing focus of the targeted 5G scenarios, the 5GCity Orchestrator needs to be able to orchestrate also a special type of VNFs, namely VNFs that are ME (Multi-access Edge) apps, according to the ETSI MEC specification. According to the ETSI specification for using MEC in NFV-based systems [10], ME apps are indeed VNFs, and therefore they can be generally managed by an NFVO-based system, e.g., OSM. However, they use special descriptors and have special requirements, and therefore some MEC components are required in order to "assist" the NFVO in the orchestration of ME apps. There are two main reasons why these components might be required (which are also the reason why we add MEC support to our NFVO):

i. Some edge-focused VNFs might benefit a lot by using the MEC descriptors instead of (or in addition to) the VNFD (Virtual Network Function Descriptor), because it includes many additional edge-related parameters such as latency characteristics.
ii. Some edge-focused VNFs might require to interact with special edge services (e.g., location services), which are modelled and operated according to the ETSI MEC specifications and interfaces.

The MEC components attached to the NFVO are explained in the following, while their interaction with the NFVO and the infrastructure layer is done in line with the related ETSI specification [10] and explained in more detail in D3.1 [11], because it is tightly coupled with the edge VIM/NFVI and with the infrastructure layer.

The MEC components required to implement the core ETSI MEC functionalities in the 5GCity system are in particular:

- Multi-access Edge Application Orchestrator (**MEAO**)
- Multi-access Edge Platform Manager – Virtual (**MEPM-V**)
- Multi-access Edge platform (ME platform)

Note that only the MEAO and the MEPM-V are actually attached to the NFVO within the 5GCity Orchestrator, while the ME platform is a "special" VNF that resides on edge hosts. These three components will interact between each other and also with the NFVO to allow the orchestrations of ME applications and services. As there are no publicly available full implementations of all these components, the project will implement the required subset of their specified functionalities. In particular, for the MEAO and MEPM-V we will be using fog05 [12], a distributed and pluggable architecture that implements the interfaces needed by these components and the control logic behind this. These components will interact with OSM and with the OSS/BSS (Operations Support System / Billing Support System), because of the MEC implementation that has been chosen by 5GCity. Fog05 was selected because it is easily pluggable and it already has an eagerly distributed architecture that makes trivial to implement solutions in which the nodes are distributed through the city, the MEAO needs also to interact with the service placement algorithms when it is time to insatiate a new ME application, because they need to work together to solve the placement problem and find the correct NFVI-PoP in which the ME app should be instantiated.

The ME platform is used for the communication between services and its PoC (Proof of Concept) can be implemented using an SDN controller that configures the communication path between different services.

Both MEAO and MEPM-V will be developed as plugins for fog05, implementing two different interfaces, the MEAO will implement the RO_plugin interface, which contains the methods needed by an orchestrator, and will validate the descriptors with the 5GCity information model, map the ME apps with the NSs in which they are part, and then send the NFV information to the NFVO. It will be also responsible for all the communication to the NFVO. The MEMP-V will be developed implementing the RM_plugin, because it needs to be able to understand the LCM (LifeCycle Management) and platform management for the ME platform. It will also interact with OSM for LCM and platform management. While the first one is a known interface of OSM, the second one will be developed based on the OSM REST (REpresentational State Transfer) API.

The interface that will be developed is the one between OSS/BSS and MEAO, that should follow the OSM REST API, for the sake of simplicity, the one between MEAO and NFVO (OSM in our case) that will use the northbound interface present in ETSI OSM, then the one between MEAO and MEMP-V can be implemented using the data centric abstraction provided by the fog05 communication model and so be an always on pub/sub communication that enable the MEAO and the MEMP-V to notify themselves when something happen. The interface between MEMP-V and VNFM regarding the LCM is not exposed by ETSI OSM so in this case has to be implemented and will follow the OSM REST API, but the PM information will come from the OSM Monitoring tool API. Then, the interface between the ME platform and MEPM-V can also be always on pub/sub, because this reference point is not defined in ETSI MEC. Please refer to Deliverable D3.1 [11] for MEC-specific architectural details. In the current context it can be practically considered as an "attachment" or "extension" of the NFVO, not interacting directly with the other components of the 5GCity Orchestrator.

Resource Placement

Resource placement is responsible for computing an optimal allocation of VNFs over different physical resource domains. The component needs to keep track of the resource usage, the edge network status, and more. Examples of input data can be the geographical positions of VMs (Virtual Machines) and available resources in terms of vCPU (virtual Computer Processing Unit), vRAM (virtual Random Access Memory), and vHDD (virtual Disk). Appendix B. investigates related work with regard to the well-investigated topic of optimal placement, and discusses the potential peculiarities of the 5GCity system. The output of the placement algorithm will be returned to the NFVO, which will enforce the respective deployment plan by interacting with the Infrastructure Abstraction.

The architecture of Resource Placement is illustrated in Figure 4 and consists of the following functional blocks:

- **Placement Algorithm:** It is responsible for running the algorithm that computes the deployment plan. Note that many different algorithms or flavours of them might co-exist (e.g., performance optimization-based, machine-learning based) and be used in different circumstances.
- **Resource Monitoring Driver:** It retrieves, checks, and organizes all the monitoring parameters that are required by the Placement Algorithm, e.g., the status and load of VMs. It performs this by using the services of the Monitoring component (see also section 2.3)

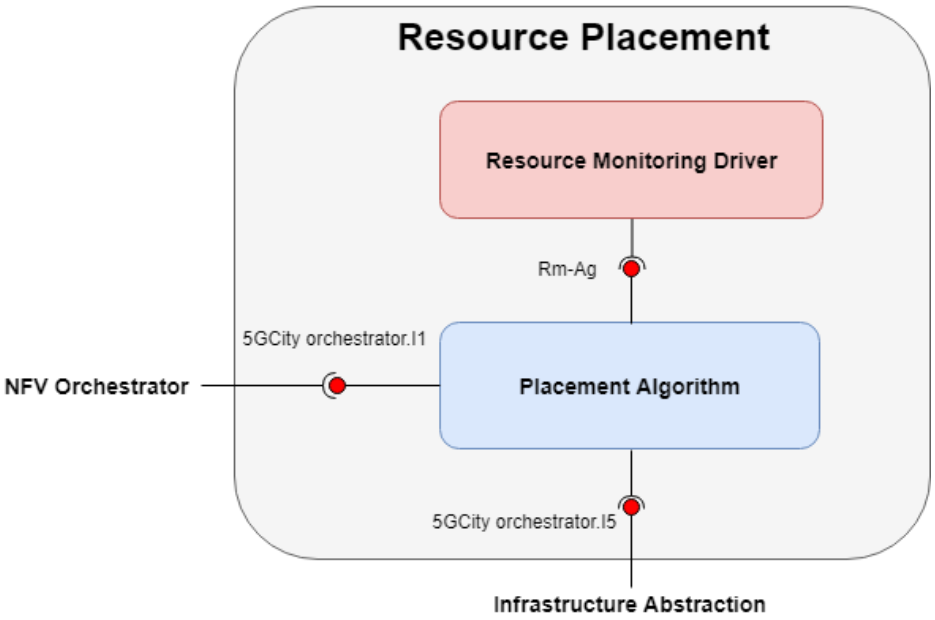

Figure 4. Internal architecture of Resource Placement.

SLA Manager

Service Level Agreements (SLA) represent the relationship between a service provider and a customer to ensure and maintain an acceptable level of Quality of Service (QoS). It includes monitoring, managing and reporting of delivered vs contracted QoS. QoS-based prioritization and low latency are considered as major points to maximize the availability and response time of network services. Thus, the 5GCity SLA Manager should be able to perform the following:

- Align the high-level end-user requirements with the low-level system parameters.
- Re-allocate resources always adapting to the system state.
- Enforce SLA for each Service Provider.

In order to achieve this, the SLA manager (Figure 5) is composed of the following main entities:

- **SLAs Repository** is where the SLAs bound to the Network Services are stored. The SLA data model is yet to be defined, but it should be aligned with data models used by the Monitoring, so that the Monitoring Driver can register alarms that involve parameters that are "shared" between the SLA Manager and the Monitoring.
- **Dispatcher** is the functional block which implements the logic of the SLA manager, triggering internal interaction (read/write of database entries), triggering API calls to NFVO to gather monitoring data, trigger an action whenever certain threshold for a specific monitored parameter is exceeded, and thus indicating an SLA violation.
- **Monitoring driver** is the entity in charge of requesting from the NFVO the creation of a performance job for the specific Network Service under analysis. This entity is also responsible for gathering the performance metrics exposed by the monitoring system via the NFVO, and is also in charge of generating an action in case of a SLA violation event.

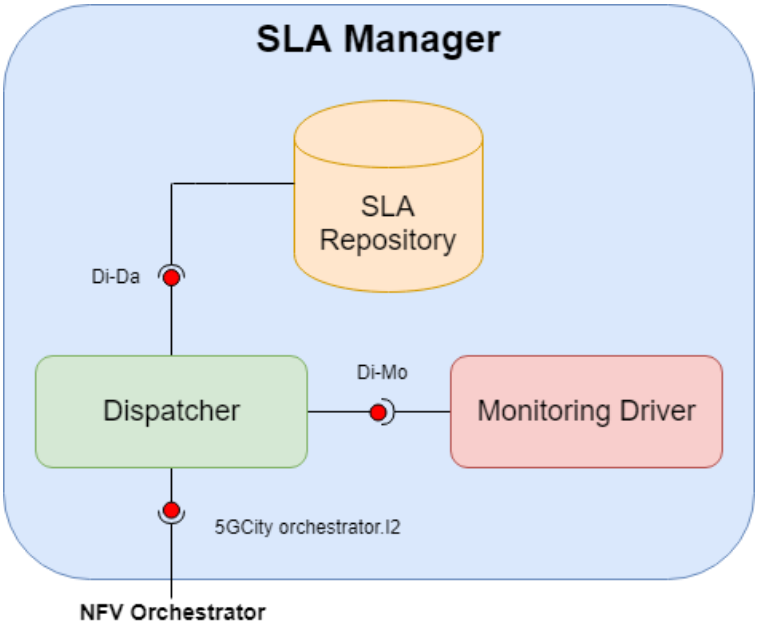

Figure 5. 5GCITY SLA manager architecture

Slice Manager

Network slicing enables the virtualized and non-virtualized network elements and functions to be easily logically segmented, configured and reused (isolated from one another) in order to meet various demands. Each slice has its own specific network architecture, mechanism and network provisioning.

Network slicing in 5GCity will cover end-to-end service provisioning, management, termination, and operation workflows on a per-slice basis. The slices will be dynamically allocated in the network and on the Multi-access edge, extending through the radio infrastructure. Network slicing will enable a unified system for dynamic deployment and provisioning of novel services, and allow the efficient and secure control and orchestration of services for different verticals. Each slice possess specific requirements related to quality policies, security functions, functions routing, radio resource management, cloud and network resource management capabilities, etc. The deployed architecture will be capable of:

- Creating customizable and user-specific slices with flexible and dynamically deployable resources, based on explicit user requests, existing slice templates, or implicit info about the planned services.
- Dynamic provisioning and instantiation of end-to-end network slices.
- Multi-tenancy and support of different underlying technologies and resources (that are used to compile slices).
- Various runtime operations and control during the entire lifetime of slices.

As shown in Figure 6, the internal architecture of the Slice Manager for the 5GCity orchestration platform consists of three main functional blocks: 1) Network Slice Life-Cycle Management, 2) Policy Management Function, and 3) Slice Repository.

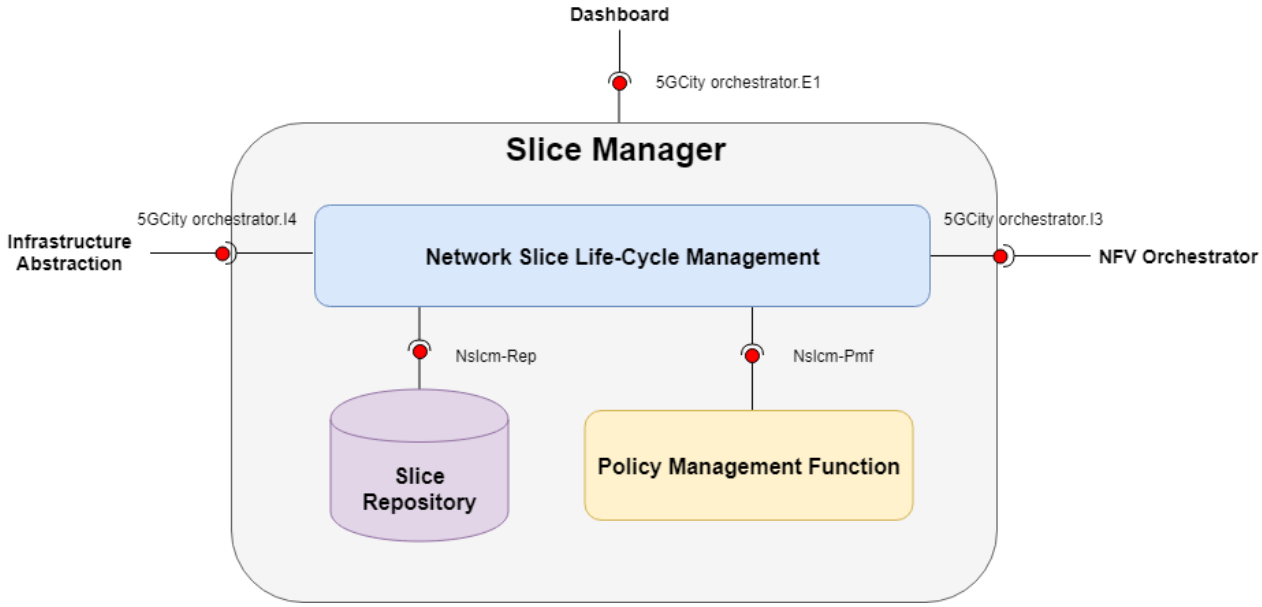

Figure 6. Internal architecture of Slice Manager.

The Slice Manager is triggered via the dashboard and is responsible for interacting with the Infrastructure Abstraction to control and manage slices over the physical and virtual infrastructures.

Network Slice Life-Cycle Management is an entry point to the Slice Manager of the 5GCity orchestration platform that can be accessed by the Slice User via the Dashboard. It is responsible for coordinating and allocating required slices and services upon request. As shown in Figure 7, with the Network Slice Life-Cycle Management, it is possible to request a slice with certain properties, deploy services, provision a network function, or modify the resources allocated to the existing slice. A Slice User can also request to terminate the slice completely and release the assigned resources. The Life-Cycle Management involves the following:

- **Slice creation** is responsible for handling requests to create slices and for interacting with the Policy Management Function and the Slice Repository to determine if it is possible to create a new slice. Note that slice creation might be done i) by passing the exact requested parameters of a slice, ii) by using existing “slice templates”, or iii) by passing information about the NSs that shall run on the template and letting the Slice Manager to craft an appropriate slice. If the slice creation is possible then the Network Slice Life-Cycle Management reserves and assigns a part of infrastructure for exclusive use of a slice owner to be isolated form other slices in the sense of traffic, security, and resource usage.
- **Slice operation** is available in order to deploy network services and provision network functions in isolated fashion for its customers. Slicing operation must be done according to the applicable policies and the deployment environment.

- **Slice maintenance** monitors performance- and security-related aspects of the slices, implicitly enabling an enhanced management of the lifecycle of the network services that run upon the slice. For example, a network slice can be scaled at runtime, adjusting the slice to the desired level of performance.
- **Slice termination** ends the life of the slice and all the NS and VNF running on it. The following aspects should be undertaken before the slice termination: i) The network services need to be stopped and deleted, ii) the involved nodes need to be released, and iii) the VIM-defined network needs to be deleted. Once the slice has been terminated, all the resources associated with the slice will have returned to the pool of available resources.

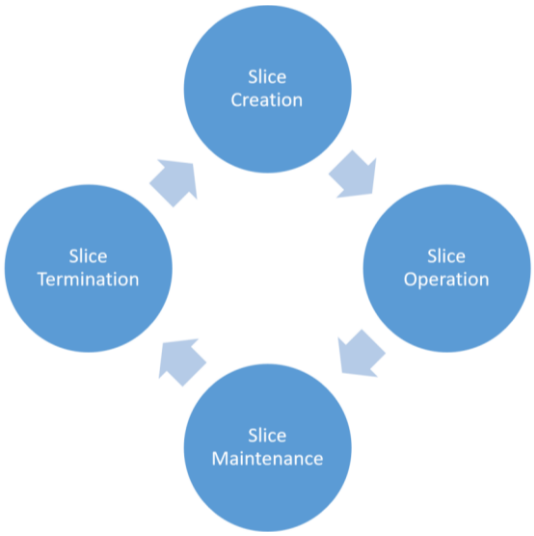

Figure 7. Network Slice Life-Cycle Management.

Slice Repository is responsible for storing information related to the network slices. It stores Information related to the virtual resources and physical resources of the slices, such as which virtual resources belong to the slice, which physical resources they are mapped to, and to whom each slice belongs. Some of the main parameters that compose a slice are the following:

- Slice id and name
- Lifetime
- Owner
- Required resources (which will be concretized and provisioned by the Infrastructure Abstraction by a series of commands that will be eventually directed to VIMs, SDN controllers, and access network controllers). These include:
    - Compute
    - Storage
    - Network
    - (RAN or WiFi) Access resources
- List of VLAN tags that identify its networking resources
- The CIDR (Classless Inter-Domain Routing) identifier(s) that is used for devices attached to its VLAN(s)

The above can be elaborated to a qualified and more complete Information Model (aka "Slice Descriptor"), which is an important missing part of the NFV world, though critical for the management of 5G services within our neutral host scenario.

Policy Management Function is responsible for runtime policies enforcement over the deployed slices such as creation and expiration dates. This module could be extended to support other specific policies such as security, etc., to be reinforced on a particular slice."

Infrastructure Abstraction

Infrastructure abstraction is the element of the 5GCity orchestrator which enables communication between 5GCity orchestration platform with the multiples underlying infrastructure elements such as VIMs (i.e. Core VIM, Edge VIM, and Extended Edge VIM) and WAN parts (i.e. SDN controller, fronthaul network and backhaul network).

Infrastructure abstraction is composed of two plugins (i.e. VIM plugin and WAN plugin) for communicating independently with the underlying infrastructure. The VIM plugin is a specific component for the VIMs implementation, which allows the 5GCity orchestrator to interact with the VIMs in a vendor- and technology-independent way, thus providing extra flexibility for configuration and implementation of different technologies used in the 5GCity slices. The VIMs supported in this platform are depicted in Figure 8 and described in below:

- **Core VIM,** which could be implemented by using OpenStack, an open source software for creating private and public clouds [13]. It is a cloud operating system that controls large pools of compute, storage and networking resources throughout a datacentre.
- **Edge VIM** is an extension of the OpenStack with performance optimization for the edge and support for Trusted Computing (see also [11]).
- **Extended Edge VIM**, which can be implemented by using Fog05, a computing VIM that runs on low-end devices [12]. It unifies the compute fabric that spans across things, edge and cloud infrastructure. In addition, Fog05 unifies administration, management and monitoring end-to-end.

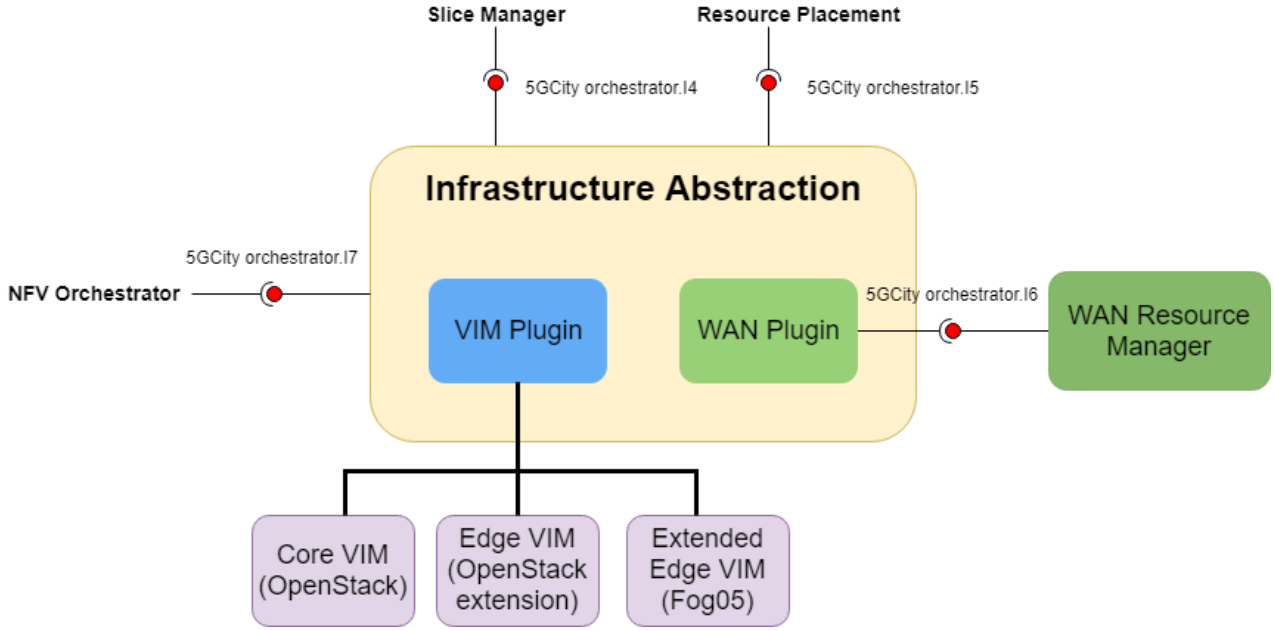

Figure 8. Infrastructure abstraction architecture.

Each of these VIMs is designed to have an interface offering specific functions with the ability to store VM images, deploy VMs, and configure their internal networking. For this, there must be a relevant database to store the images needed for the service deployment. The aforementioned VIM plugins will allow 5GCity

platform to continuously expand and support new VIM implementations or follow the evolution of the already supported ones.

The WAN plugin is the second component of the infrastructure abstraction, which exposes an interface to the WAN Resource Manager element for dealing with the underlying network such as SDN controller, fronthaul and backhaul networks. Thus, it allows WAN Resource Manager to control the WAN networks, e.g. be able to define the data traffic flows. In this plugin, the functionalities greatly depend on the underlying networking technologies employed as well as on their virtualization capabilities.

To determine whether there is an existing open-source solution suitable to be a basis for the role of the 5GCity Edge VIM, an investigation was done considering OpenStack and OpenVIM. OpenStack, being a cloud operating system based on an open-source software with an active community and OpenVIM, introduced as a lightweight VIM part of ETSI Open Source MANO (ETSI OSM) [9].

The results of the benchmark and analysis of the two solutions, presented in detail in Deliverable D2.2 (section 3.7.2), showed advantage for OpenVIM in terms of performance, however this comes at the cost of reduced functionality and flexibility. On the other side, the more complex and general-purpose OpenStack has an active community and allows for the creation of custom solutions for the need of the 5GCity Edge VIM.

In 5GCity an Edge VIM will be provided based on OpenStack with a Trusted Computing Pools feature enabled for edge devices which have been described in Deliverable D3.1 [11] together with the related extensions that will be developed at the NFVI level. Relying on hardware-based security, combined with an attestation server, the compute nodes will be running only trusted software with verified measurements. Today's implementation of Trusted Computing Pools is based entirely on the Intel TXT technology while this OpenStack feature will be extended to support ARM devices with built-in ARM TrustZone technology.

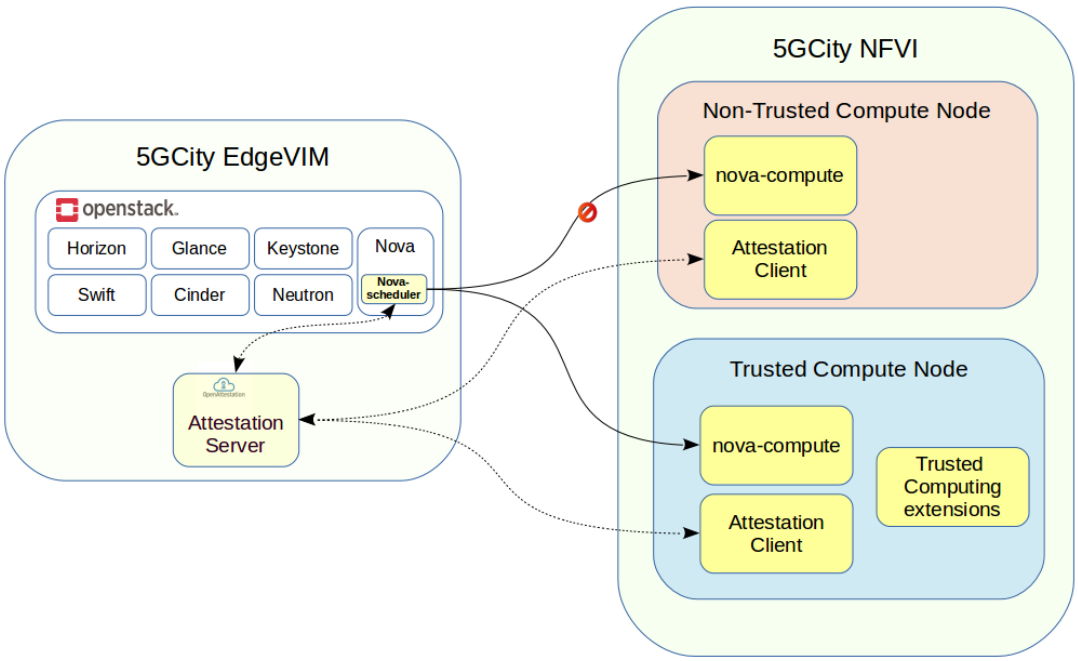

Figure 9. Edge VIM simplified architecture

The simplified architecture of the Edge VIM as well as an overview of the compute node verification mechanism are shown in Figure 9. The nova-scheduler service, part of OpenStack Compute, requests

information from an attestation server whether the compute node is trusted or not. The attestation server on its turn challenges the compute node and verifies the received data against a whitelist database to determine node trustworthiness. The detailed architecture and description of the EdgeVIM can be found in Deliverable D3.1 [11].

WAN Resource Manger

The WAN Resource Manager is responsible for managing resource of the lower layers, and especially for configuring the traffic flows among them. The simple usage of an SDN Controller such as ODL (OpenDayLight) is being considered, and therefore the WAN Resource Manager would simply use Northbound API of that controller to set the flows, as well as some information from the Infrastructure Abstraction about the underlying infrastructure technologies, in order to decide how to set the flows and build up appropriate WAN configurations.

2.1.4. Main orchestration interfaces

The 5GCity orchestrator is designed as a set of components, interconnected via external and internal interfaces. The external interfaces connect the 5GCity orchestrator with the rest of the 5GCity platform, while the internal interfaces refer to the internal interactions among the different entities composing the orchestrator.

NOTES:

1. *The present documentation of the basic REST operations is a high-level design output that will guide the development of the interfaces. The full list of operations and their fully-fledged documentation shall be created with development tools during the development of the prototype, but always based on the scope and the structure indicated here.*
2. *The descriptions of the interfaces use (in their parameter lists) high-level objects such as ns, vnf, sla, and slice. These objects will be implemented based on the respective data models. For ns and vnf, these data models will be based on versions or extensions of the respective ETSI NFV descriptors, namely nsd and vnfd, as specified in* [14] *and* [15]*, respectively. For sla and slice, they will be based on similar descriptors, which will be developed by the 5GCity project upon the basis of the parameters and the functionality described for the SLA Manager and the Slice Manager in section 2.1.3.*

The 5GCity orchestrator interfaces as depicted in Figure 3 are external and internal.

EXTERNAL INTERFACES

5GCity orchestrator.E1 is the external interface offered by the 5GCity orchestrator and used by the Dashboard, in order to request the creation/onboarding and manipulation of slices, NSs, VNFs, and SLAs. For example, Table 1 documents the planned REST operations for acting upon slices.

Operation Type	Resource	Parameters	Metadata	Description	Expected Result
GET	orche1/slice	slice_id	Authentication Data	Retrieve the slice identified by "slice_id"	All data and metadata related to the slice with "slice_id"
POST	orche1/slice	slice	Authentication Data	Request the setup of a slice with the provided parameters	STATUS (SUCCESS or one of the to-be-defined ERROR CODES)
PUT	orche1/slice	slice	Authentication Data	Request the update of a slice with the provided parameters	STATUS (SUCCESS or one of the to-be-defined ERROR CODES)on

DELETE	orche1/slice	slice_id	Authentication Data	Request the deletion of the slice identified by "slice_id"	STATUS (SUCCESS or one of the to-be-defined ERROR CODES)
POST	orche1/compileSlice	ns[], vnf[], sla[]	Authentication Data	Request the compilation of a slice that fits the provided parameters	All data and metadata related to the slice compiled by the intelligence of the slice manager

Table 1. 5GCity orchestration.E1 interface description

Note that while the *orche1/slice* operation "tells" the Slice Manager what to do with slices, *orchi1/compileSlice* also tells the Slice Manager to create a slice, but provides only indications of what shall run upon the slice, leaving the details of the slice up to the intelligence of the slice manager.

Similar REST resources exist within this interface for the other main managed entities:

- *orche1/ns*
- *orche1/vnf*
- *orche1/sla*

Note that while orche1/slice concerns the slice manager, orche1/sla is directed to the SLA manager and the two others (orche1/ns and orche1/vnf) are handled by the NFVO.

5GCity orchestrator.E2 is the external interface offered by the Monitoring and used by the 5GCity orchestrator, in order to request or subscribe to orchestration-critical monitoring values.

The current intention is that this interface correspond with a simple usage of the API that is offered by the Monitoring module as a service not only to the orchestrator, but also to all modules that exploit monitoring information. Therefore, the details belong to the development of the Monitoring module itself (see also section 2.3). Here it is just noted that the main usage of this interface by the 5GCity orchestrator will be for:

- Subscribing (from the SLA manager) to monitoring parameter values, which should cause the SLA manger to trigger a reaction if certain thresholds are exceeded.
- Collecting of related performance characteristics (mainly from the NFVO and the Resource Placement) whenever they need to run optimizations (with regard to the configuration and allocation of network services, VNFs, and slice resources).

5GCity orchestrator.E3 is the external interface offered by AAA and used by the 5GCity orchestrator, in order to allow access to the system via validation of the users.

Again here, this interface corresponds with simple usage of the (AAA) interface, which will be available for all components that require authentication, especially the Dashboard. The main usage of this interface within the 5GCity orchestrator will be for:

- Identifying and authenticating the user when interaction between different orchestrator components are performed, especially when these components reside in different administrative domains.
- Checking the authorization of a component (for a given user/role) to trigger certain tasks implemented by a different component (e.g. slicing) by calling the interfaces of the latter.

INTERNAL INTERFACES

5GCity orchestrator.I1 is the internal interface offered by the Resource Placement and used by the NFV Orchestration, in order to trigger the decision process for finding an optimal placement of all the resources involved in a network service. Table 2 documents the main operations for this functionality.

Operation Type	Resource	Parameters	Metadata	Description	Expected Result

POST	orchi1/computeDeploymentPlan	ns, vnf[], slice, sla	Authentication Data	Retrieve an optimal placement/deployment plan for the provided ns/vnfs with the provided sla on the indicated slice	The deployment plan to be enforced (by the Infrastructure Abstraction, which will receive it and pass it over to the appropriate VIMs as the appropriate set of commands)
GET	orchi1/activateAlgorithm	algorithm_id	Authentication Data	activate/select one of the placement optimization algorithms that can be used by the Resource Placement module	STATUS (SUCCESS or one of the to-be-defined ERROR CODES)

Table 2. 5GCity orchestration.I1 interface description

5GCity orchestrator.I2 is the internal interface offered by the SLA Manager and used by the NFV Orchestrator, in order to associate a network service with an SLA, which will be the basis upon which the SLA manager will keep monitoring thresholds and identifying violations. Such violations are also accessible for the NFV Orchestrator via operations of this interfaces. The main operations are described in Table 3.

Operation Type	Resource	Parameters	Metadata	Description	Expected Result
GET	orchi2/sla	sla_id	Authentication Data	Retrieve an SLA identified by "sla_id"	All data and metadata of this SLA (including information about the service it is associated with)
POST	orchi2/sla	sla, ns_id	Authentication Data	Register an SLA to be stored by the SLA manager, providing info about the network service it is associated with	STATUS (SUCCESS or one of the to-be-defined ERROR CODES)
PUT	orchi2/sla	sla, ns_id	Authentication Data	Update an SLA to be stored by the SLA manager, providing info about the network service it is associated with	STATUS (SUCCESS or one of the to-be-defined ERROR CODES)
DELETE	orchi2/sla	sla_id	Authentication Data	Delete an SLA identified by "sla_id"	STATUS (SUCCESS or one of the to-be-defined ERROR CODES)
GET	orchi2/violation	sla_id	Authentication Data	Retrieve a violation (identified by the SLA Manager) related to a specific SLA, identified by "sla_id"	All data and metadata of the identified violation

Table 3. 5GCity orchestration.I2 interface description

Note that the retrieval of violations by the NFVO might be done periodically or triggered by an alarm or "push" operation, but this is an implementation detail which is out of scope at this point.

5GCity orchestrator.I3 is the internal interface offered by the Slice Manager and used by the NFV Orchestrator, in order to check the ownership and other details of a slice before initiating NS- and VNF-related orchestration actions that contradict these details. The main operation is described in Table 4.

Operation Type	Resource	Parameters	Metadata	Description	Expected Result
GET	orchi3/slice	slice_id	Authentication Data	Retrieve slice identified by "slice_id"	All data and metadata related to the slice with "slice_id"

Table 4. 5GCity orchestration.I3 interface description

5GCity orchestrator.I4 is the internal interface offered by the Infrastructure Abstraction and used by the Slice Manager, in order to enable in a VIM-technology-independent way the triggering of actions that are required to actually put a slice in place. The main operations are described in Table 5.

Operation Type	Resource	Parameters	Metadata	Description	Expected Result
POST	orchi4/allocateResource	resourceType, resourceData, owner	Authentication Data	The type (e.g., RAN/WiFi capacity, network capacity, Compute node, storage space) and the details of the	A "resource_id" in case of SUCCESS or one of the to-be-defined ERROR CODES

				required resource, along with an owner who shall also be passed on to the VIM	
POST	orchi4/freeResource	resource_id	Authentication Data	Free the slice identified by "slice_id" by calling the VIMs that are responsible for the elements of the slice	STATUS (SUCCESS or one of the to-be-defined ERROR CODES)

Table 5. 5GCity orchestration.I4 interface description

Note that a sequence of various "orchi4/allocateResource" invocations will be needed in order to "set up" a slice. These will have in turn to translated by the Infrastructure Abstraction to VIM-specific commands.

5GCity orchestrator.I5 is the internal interface offered by the Infrastructure Abstraction and used by the Resource Placement, in order to retrieve information needed in order to compute an (optimal) placement/deployment plan, i.e. one that have been computed by an execution of the *orchi1/placement* operation. The main operations are described in Table 6.

Operation Type	Resource	Parameters	Metadata	Description	Expected Result
GET	orchi5/status	resource_id	Authentication Data	Retrieve the status of the resource identified by "resource_id"	Data and metadata about the status of the resource identified by "resource_id"

Table 6. 5GCity orchestration.I5 interface description

5GCity orchestrator.I6 is the internal interface offered by the Infrastructure Abstraction and used by the WAN Resource Manager. It will be used for simple information gathering (about the infrastructure) and its design will also depend on the SDN controller that the WAN Resource Manager will control. It will be used to setting up traffic flows with conventional usage of SDN controllers, and no significant extensions to the existing solutions are planned.

5GCity orchestrator.I7 is the internal interface offered by the Infrastructure Abstraction and used by the NFV Orchestrator to deploy NSs and VNFs. This should be implemented in accordance with the "API Service and Utilities" of the "Resource Orchestrator" of OSM. Some extensions might be dictated by the fact that our Infrastructure Abstraction will support a broader palette of underlying VIMs, namely edge- and extended edge-related VIMs in addition to standard Cloud VIMs such as OpenStack, as explained previously in section 2.1.3. Table 7 shows some of the involved operations.

Operation Type	Resource	Parameters	Metadata	Description	Expected Result
GET	orchi7/status	resource_id	Authentication Data	Retrieve the status of the resource identified by "resource_id"	Data and metadata about the status of the resource identified by "resource_id"
POST	orchi7/deploy	vnf, resource	Authentication Data	Deploy the provided VNF to the specified resource	STATUS (SUCCESS or one of the to-be-defined ERROR CODES)
…	..	…	…	(other operations that will be generic and technology-independent versions of deployment-related actions that can be performed by the used VIMs)	…

Table 7. 5GCity orchestration.I7 interface description

2.1.5. Main orchestration interactions

This section describes important end-to-end workflows for the 5GCity orchestrator platform.

Resource Placement

As depicted in Figure 10, requests for deploying a service trigger a resource placement workflow, which computes the optimal placement of the involved VNFs. This can be done by using with the NFVO and the Infrastructure Abstraction to interact with the resources that can act as VNF hosts. The whole process is assisted by a Monitoring Driver, which delivers also dynamic, performance-related information about the resources.

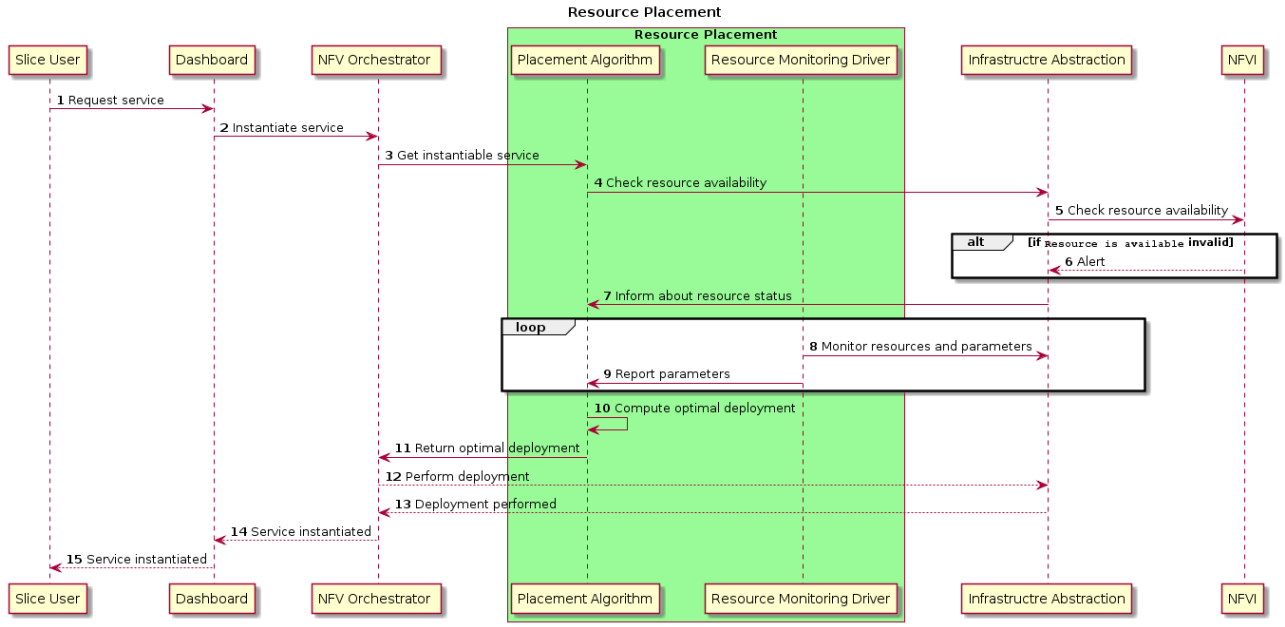

Figure 10. Resource placement workflows

SLA Manager

The overall logic which describes the interaction between SLA manager and the rest of 5GCity architecture upon the instantiation of a new service that is accompanied by an SLA is described in Figure 11.

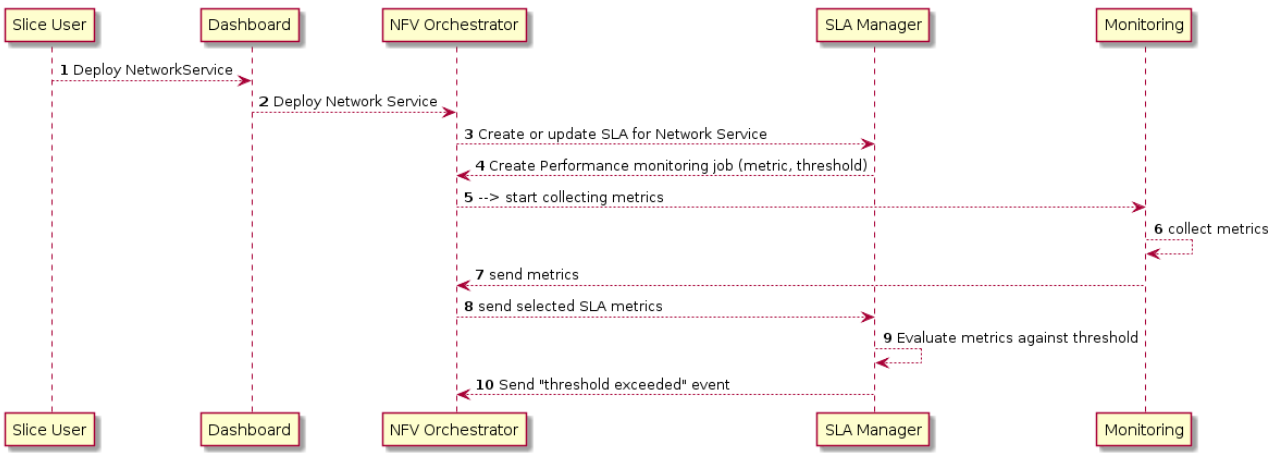

Figure 11. SLA manager sequence diagram

Slice Manager

This section describes workflows of the Slice Manager from the point of view of the Slice User. The scenarios include creating a new network slice, executing an operation upon a slice, maintaining/monitoring the slice, and terminating a deployed network slice.

Slice Creation

Figure 12 shows how 5GCity slices are created in the 5GCity orchestration platform. In principle, when a Slice User requests a slice, the request is forwarded to the Slice Manager (namely to its Network Slice Life-Cycle Management component). This component is responsible for checking the slice policy with the Policy Management Function prior to the creation of the slice. Once the Policy Management Function module approves the creation process based on the existing policy then Network Slice Life-Cycle Management needs to check with the network slice repository. If there is no slice with conflicting data, then the Network Slice Life-Cycle Management creates the requested slice and adds the information to the repository.

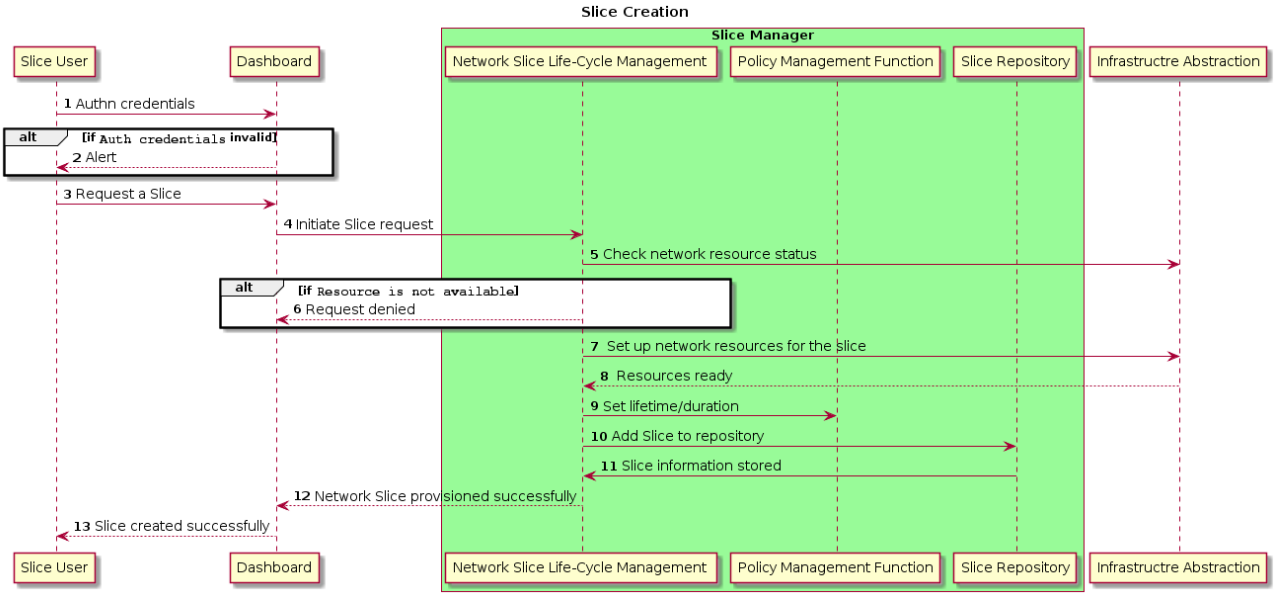

Figure 12. Network slice creation internal process.

Slice Operation

Once a network slice is created, it becomes ready for assigning NSs and VNFs to it. A Slice User can access the Slice Manager via the 5GCity Dashboard and request for a specific slice. Upon access to the slice, the Slice User is able to perform various actions, e.g., fetch Network Service Descriptors (NSD) and Virtualized Network Function Descriptors (VNFD) from the catalogue in order to create a service and ask for its deployment upon this specific slice.

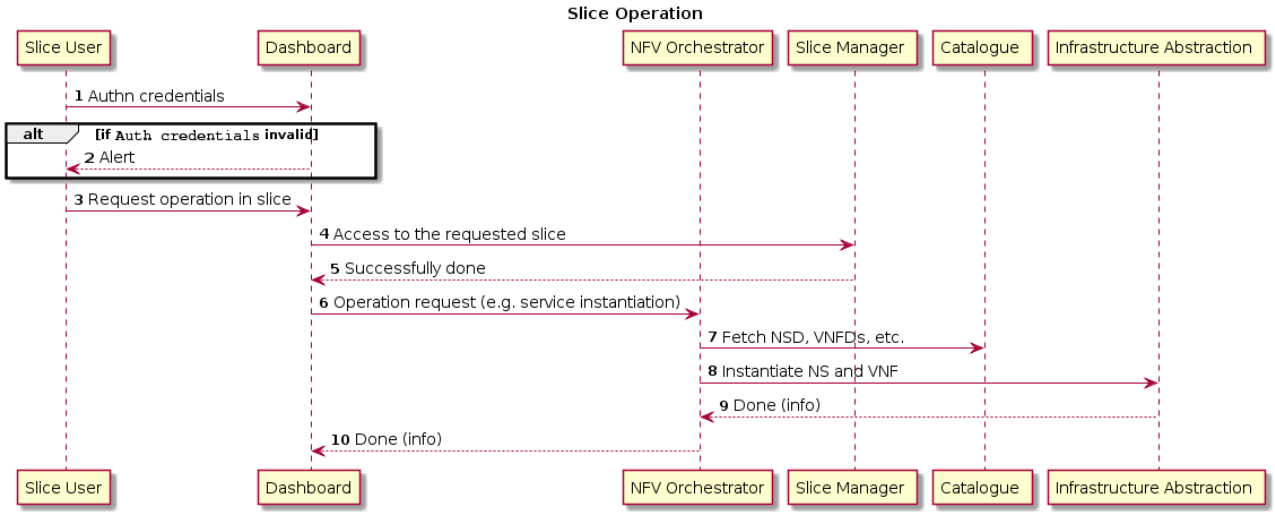

Figure 13. Network slice operation.

Slice Maintenance

Figure 14 depicts the workflow of monitoring/maintenance actions upon a slice. The Slice User logs and authenticates again over the 5GCity Dashboard and, having access to the slice, she/he is allowed to perform certain maintenance actions upon the slice. These differ from the slice operation actions in that they are related to the slice itself rather than to what is running upon the slice.

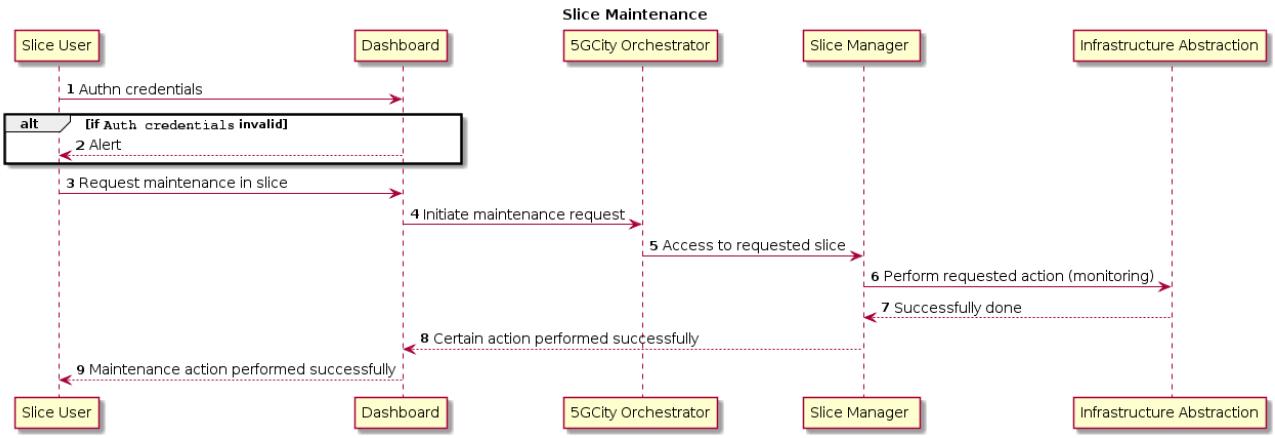

Figure 14. Network slice maintenance.

Slice Termination

A slice can be terminated manually by the Slice User or automatically after its lifetime period. Upon termination of a network slice, the NSs running on it are stopped and the resources returned to the available network resources. Finally, the information related to the network slice will be deleted from the repository. The slice termination workflow is shown in Figure 15.

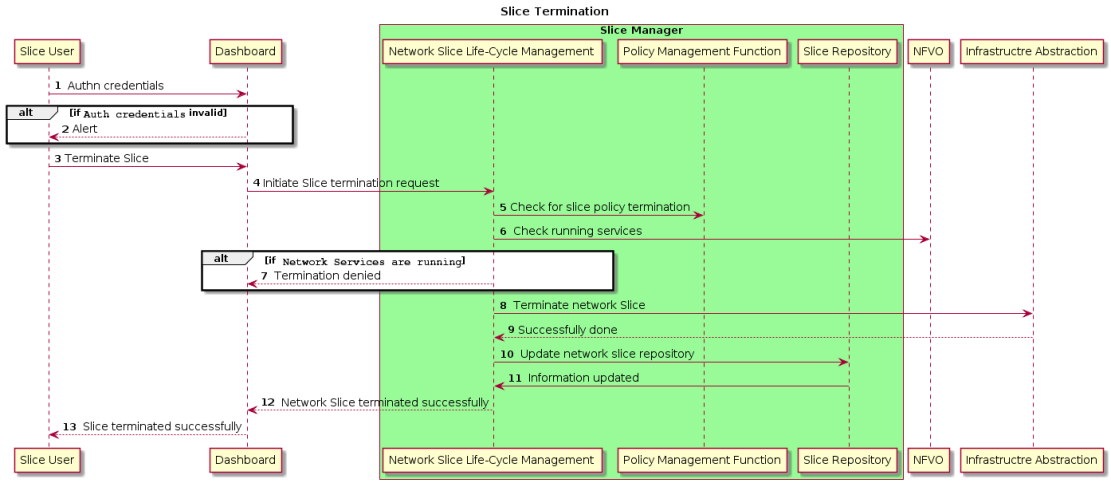

Figure 15. Network slice termination.

2.2. Dashboard

5GCity's dashboard is aimed to be the main interface to the majority of 5GCity's stakeholders. As depicted in Figure 16, Dashboard is envisioned to be 5GCity's "top" component functioning as the single-entry point for 5GCity's platform end users as well as third party applications aiming to interact with the system.

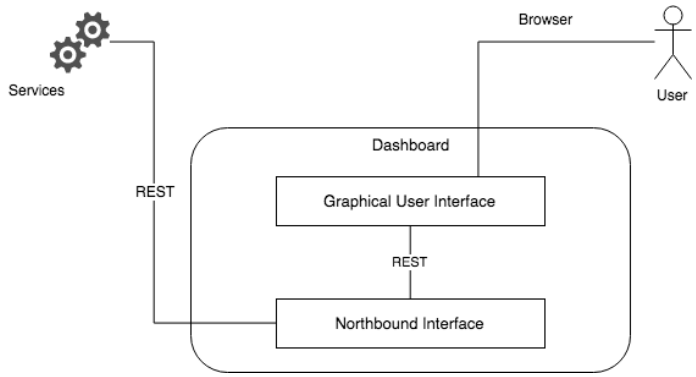

Figure 16. Dashboard High-Level Architecture

This allows to check that the Dashboard is aimed to be composed by two main sub-components:

- **Graphical User Interface (GUI)**
  The GUI will be the interface provided to 5GCity's platform end users exposing its features through an appealing and intuitive interface. The graphical user interface will provide different graphical environments based on the user's permission role. The graphical interface will be able to adapt its contents (visualised information) as well as enabled features targeting the action scope of each 5GCity role. The different roles envisioned to be supported by the GUI as well as its actions will be presented in a section below.

- **Northbound Interface (NBI)**
  This subcomponent is envisioned to provide a REST interface to 5GCity's platform third parties (including the GUI) enabling the access to 5GCity's information and features to authorised parties. This subcomponent will provide a complete overview of 5GCIty's features and information. To achieve so, it will interact with different 5GCity's subcomponents in order to retrieve the requested information correlating multiple sources of data, if needed.

2.2.1. 5GCity Dashboard roles

5GCity dashboard will have to accommodate different user roles with considerably different scopes and therefore a differentiated set of features provided to each one of them. Below, each role is presented with a minor description as well as the main set of features envisioned to be provided by the dashboard.

- **Service Developer**

  Description: This role is associated NFV and MEC app providers that will ultimately will enable the creation of added value end to end services for end users. This type of user also includes entities who create applications that can benefit from the 5GCity architecture, like for example video analytics or augmented reality software.

  Envisioned actions: Users with this role, will interact with 5GCity platform in order to manage the NFV and MEC apps which they are responsible for. Furthermore, users with this role should be able to test and monitor developed solutions in order to provide secure and optimised software.

- **Neutral Host**

Description: These entities own the necessary infrastructure that can be used for hosting the computing, storage and networking infrastructure. The available resources include space in street cabinets, lampposts or buildings along with power supply and connectivity facilities necessary to power and interconnect the ICT equipment. Users with this role can be regarded as the owners of a 5GCity platform instance and will be seen as Administrators of the platform.

Envisioned actions: Users with this role will have access to platform management features allowing an effective management of its information and access. One of the most important features to be provided to this user is the possibility to view and manage all the infrastructure registered in the 5GCity platform in a georeferenced way. Despite this physical resource management, virtual resource management is also one of the most important features to be enabled to this user. Users with this role will be responsible to accept and manage slice requests performed by Content/Service Providers and therefore will have to be able to see resource usage at both physical and virtual levels as well as monitoring information of 5GCity platform resources.

- **Slice Requester/User**

  Description: It includes entities not owning network deployments/resources but that will use virtual resources to create and provide their own end to end services. By taking advantage of the 5GCity architecture and the Neutral Host concept, these users will have a pool of different type of resources that can be selected and used to develop new and innovative services.

  Envisioned actions: One of the most important features to be provided to the current user is the ability to define and request a slice (to be accepted/rejected by a Infrastructure Owner). These users will be the responsible for the instantiation of end to end services (always in the scope of a slice which they own) so features allowing the instantiation and management of end to end services should be present in the dashboard. Furthermore, monitoring information regarding the instantiated services should also be available for these users allowing the application of remediation actions if a given SLA is not being met.

2.2.2. Graphical User Interface design and usability

The design process was split in 5 smaller and trackable phases (Figure 17). These phases are composed of:

1. a requirements definition,
2. research,
3. low fidelity User interface (UI) design and wireframing
4. high fidelity UI design and
5. Style guide (Not yet started)

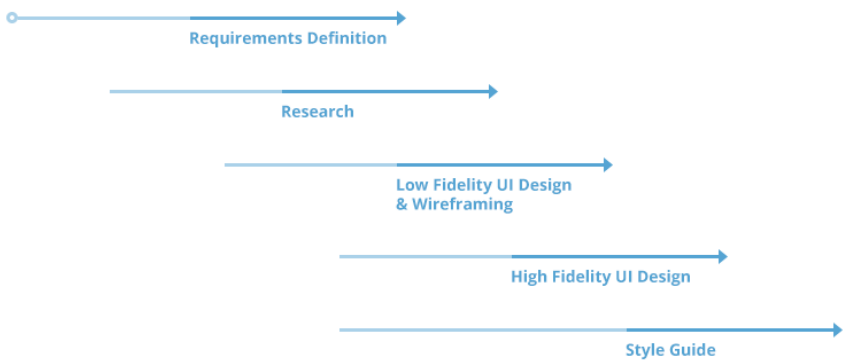

Figure 17. Dashboard design process phases

2.2.2.1.Graphical User Interface design and usability

5GCity's platform is intended to be mostly used by network tech related people, the type of users that tend to interact with systems using rudimentary, complex and unpleasant interfaces or even with a command line interface. The requirements definitions established the guidelines to follow throughout the design process of 5GCity's dashboard with the goal to break with this "Status Quo" and deliver a better and enhanced experience to end users. To do so, the following requirements for the UI were defined:

- Clean
- Appealing
- Organized
- Users should not need to read a manual or extensive documentation to be able to use its core functionalities. Everything should be as simple and self-explanatory as possible
- Follow the brand identity in terms of colour, typography and general style (look & feel)

2.2.2.2.Research

The first step of the research phase was the creation of a "mood board", i.e. a digital collage/collection of different elements that would serve as inspiration for the following phases. This methodology is useful to share and discuss different visions, the general look & feel of what is intended and how things will work in the end. Some of the main contents of this study considered relevant towards shaping important decisions for the 5GCity Dashboard appearance are provided below.

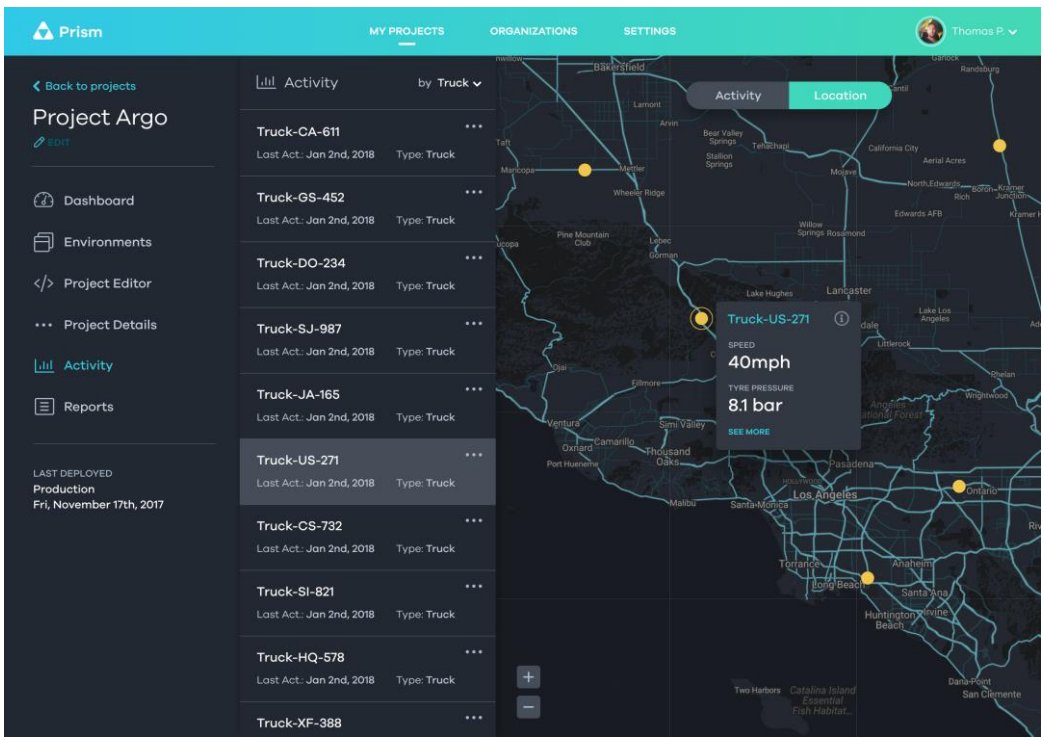

Figure 18. Example of map and list view side by side

A hybrid view with a list and map representations (Figure 18) was one the main researched topics. For example, we found this to be really helpful for the design of the "Slice Management screen" where the users need to see a list of slices along with a visual geo-representation of the slice.

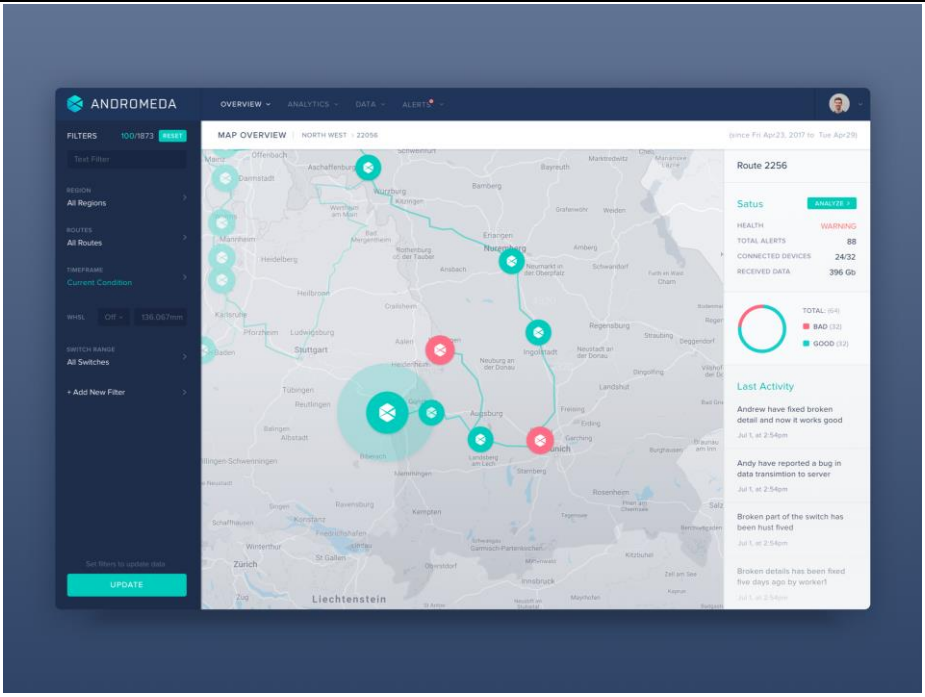

Figure 19. Example of Geo-representation

Some research was also made about the best ways to represent paths and markers on a map as well as item states and visual warnings (see Figure 19).

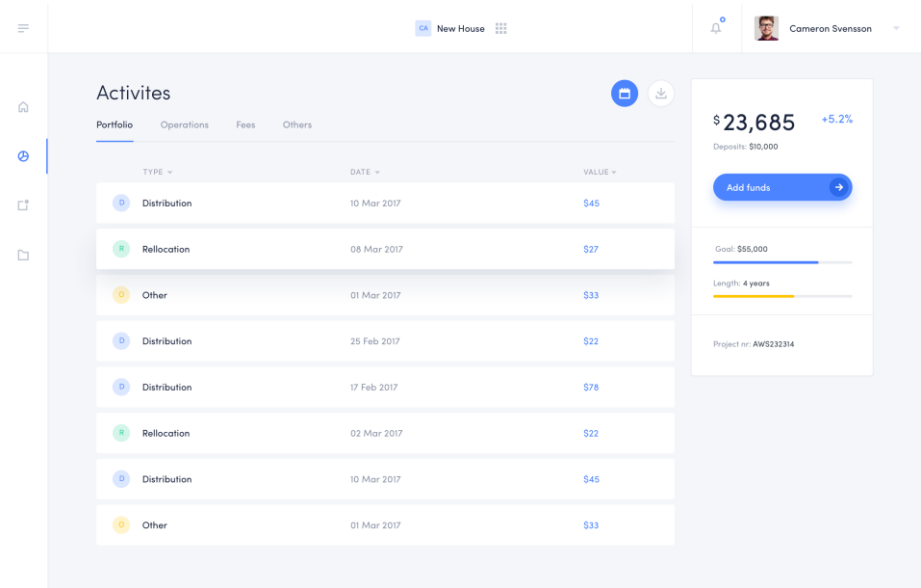

Figure 20. Example of Listings

List views (see Figure 20) also belonged to the researched topics. They will be one of the main information views of 5G City, since we considered them to be visually appealing and easy for the user (i.e., easy to understand and keep track). This also supports scalability.

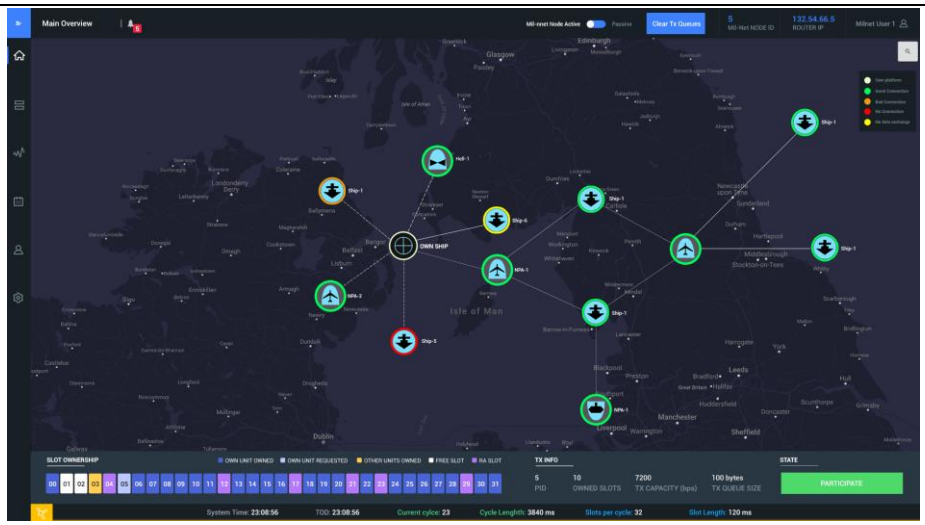

Figure 21. Example of Geo-representation with dark UI

We figured out that a dark UI (e.g., as in Figure 21) might be the best approach for our use cases, since choosing an appropriate background helped us become more efficient, which can become a key factor for a global design solution in terms of UI layout and functionality. In our experience, the use of a dark background favoured legibility, contrast, and the visual perception of item states (warnings, errors, etc.).

2.2.2.3.Low fidelity UI design

After the requirements and research phases, low-fidelity design process was started. Low-fidelity design process phase is based on drawing solutions directly on paper for the key challenges of 5GCity, the basic interface grid and structure, dashboard elements and resource management, usage and allocation.

The main goal was to diverge, to draw as many variations as possible. When drawing things in a hurry, no masterpiece is expected, it's all about the visual solution, the thinking behind it and not how it looks in the paper. Some of the outcomes of this phase are exemplified below:

- Solution will provide a left menu containing all the possible activities to be engaged by a user (based on its role).
- Information will be displayed in a minimalistic way (as collapsed as possible) allowing the user to expand details of the elements he wishes to analyse.
- Table views will be commonly adopted by the platform and will share a unified design.
- Infrastructure topology information will be contextualised with geolocation information.

2.2.2.4.High fidelity UI design

High Fidelity UI design takes into account all the gathered feedback until this state and establishes a workflow allowing to work on high fidelity mock-ups. Regarding styling, this phase takes into consideration the basic styles initially defined as well as the style guide rules evolution. This definition is currently taking place with the aim to specific in high detail 5GCity's graphical user interface. However, an example of such a specification is provided below already addressing some of the differences associated with information visualisation when comparing what users with different roles can see.

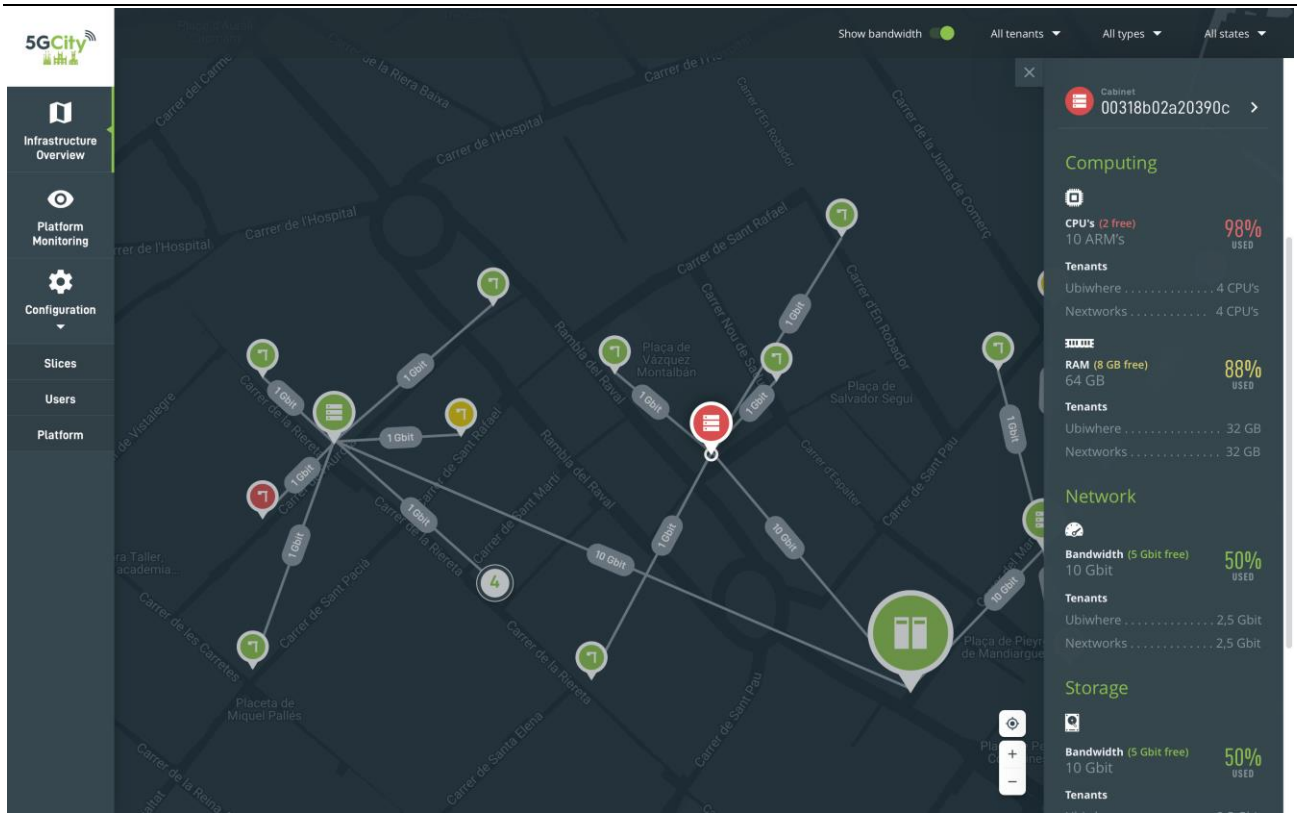

Figure 22. 5GCity Platform Infrastructure Overview available for Neutral Host role

Figure 22 already exemplifies one of the features aimed to be provided to users with the "Neutral Host" role. In this screen, users will be able to check all the infrastructure currently registered in the platform and available to be rented by Slice Requesters/Users. Per infrastructure node, users will be able to also see which entities are currently renting its resources. Infrastructure usage information is also highlighted in this screen allowing these users to check the need of expanding the infrastructure registered in the platform. Furthermore, as the figure displays, left menu options will enable users with this role to manage 5GCity platform, both in what regards its resources (physical and virtual) as well as its users.

Slice Requesters/Users will have a similar feature as the previously exemplified, containing, however, a limited set of information. Figure 23 illustrates this behaviour:

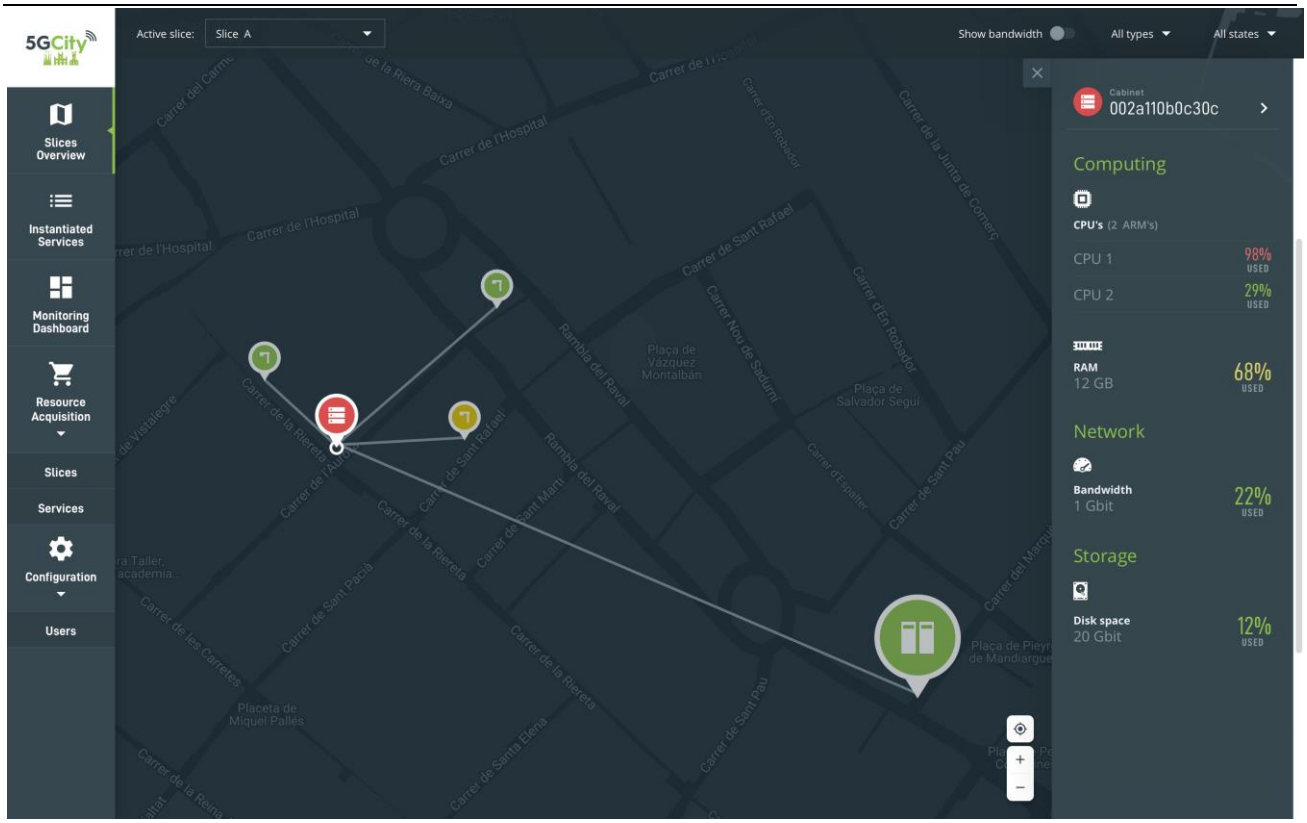

Figure 23. Slice resource visualization

As illustrated in Figure 23, the information provided to users with Slice Requester/User role will be considerably limited when compared to the similar view available for Neutral Host users. Slice Requester/User users, will only be able to check the status and topology of the virtual resources that they have previously rented being unaware of the resources rented by users with a similar role even if they share virtual resources under the same physical host. Furthermore, and also visible in the previous figure, the options provided to these users through the left panel are centred in the management of its slices and end to end services being the user agnostic of other users with a similar role that is also using the platform (using a multi-tenancy rational).

2.2.3. Dashboard implementation technologies

Dashboard implementation aims to leverage on the outcomes of EU projects SELFNET, CHARISMA and SONATA to create a state of the art Dashboard with the main outcomes of each one of the previously mentioned projects. As a consequence, components from each project are envisioned to be adapted and integrated in 5GCity platform.

In what respects authorisation and authentication methodologies, Dashboard will rely on 5GCity's Authentication, Authorization and Accounting (AAA) framework presented in section 2.4.

The technologies aimed to be used to implement 5GCity's GUI are all standard in Web development. At the basic level Hyper Text Markup Language (HTML), Cascading Style Sheets (CSS) and Javascript make up the application's foundation. On top of these standard languages, various frameworks will be used for efficiency, productivity and better data handling. For visual elements styling, it was chosen Sass, being an extension to CSS that improves on the language with new methods and efficient code structure. As for Javascript frameworks, AngularJS is aimed to be used essential for data handling and data visualization. Addressing each of them individually, AngularJS has great support and an extensive community behind the framework and thus making it one of the most used technologies for web applications development. As for

communication, the standard REST communication for the great majority of API access requests is envisioned to be used and eventually Websockets for updating in real time displayed information.

Focusing on the NBI and taking into consideration that REST is the most used technology in Web services, its contents and features are envisioned to be served also using this technology. Focusing on implementation, Python is the chosen language to develop both a REST client (that will query 5GCity's bottom layers) and a REST server that will expose an API to the GUI and other third parties. As a rest client Python library "Requests: HTTP for Humans" will be used allowing the execution of HTTP requests in a simple format while not requiring extra steps for its completion. This library can handle query strings automatically as well as POST forms. The REST server will be powered by Falcon, a high performance Python framework to build APIs. This framework follows REST standards and, was developed to be light, fast and flexible. To promote and ease the usage of the APIs offered by 5GCity, Swagger documentation will be provided.

2.3. Monitoring

The main components to be considered in the monitoring system of the 5GCity platform are the monitoring functionalities related to the overall virtualized resources (compute, storage and network) of the three-tier architecture, as well as a set of parameters related to applications and services running on the 5GCity infrastructure.

The group of infrastructure components includes three different domains of resources:

- **NFV Infrastructure (NFVI) resources** that comprise compute, network and storage virtual resources;
- **SDN-enabled elements**, including physical and virtual resources, which are usually controlled by a SDN controller;
- **Physical devices** that do not belong to the previous categories, such as non-SDN compliant network routers and switches, Small Cells, PNFs and other devices for which we are interested in collecting monitoring information.

The second group of functionalities includes:

- **Virtual Network Functions (VNFs**) virtual machines performing specific network functionalities;
- **Service monitoring parameters** that represent metrics; they are tracked (meaning data collection) to check the level of compliance of a specific running service with the agreed SLAs.

The monitoring system should be able to be integrated with several orchestration layer components, such as the Resource Manager, the Slice Manager, the SLA Manager, and the OSS/BSS systems to provide decision support for multiple purposes such as security threat detection and mitigation, SLA assurance, resource optimization and root cause analysis.

2.3.1. NFVI monitoring

The 5GCity architecture integrates the Cloud and Edge concepts and it is based on a three-tier model, which reflects three different geographical areas. 5GCity resources are deployed and identified with: (i) a centralized area where massive computing resource are deployed, (ii) an edge area with limited computing resources, corresponding to street cabinets, and (iii) an edge area with wireless devices. The resulting underlying NFVI infrastructure includes resources distributed along these three levels as well as the back-haul and the front-haul network.

It is important to keep track of the key performance metrics within the 5GCity, distributed infrastructure, in particular:

- CPU, Ram and Hard Disk utilisation: especially on the MEC nodes, belonging to cabinets and lampposts, where it is important to monitor the resources allocation and possibly keep it as low as possible;
- Virtual network devices utilisation, i.e. bitrate on virtual link, packets metrics on data flows, etc.
- System physical resources, i.e. Radio resources, LTE and Wi-Fi, PNF (Physical Network Functions).

2.3.2. Applications and Services monitoring

VNFs

These parameters can be either VM-related information (e.g. CPU utilisation, bandwidth consumption) or VNF specific such as, calls per second, number of subscribers, *number of rules, flows per second*, VNF downtime, etc. One or more of these parameters, depending on the implemented logics, could also trigger a reaction on the QoS loop.

Services

At Services level, monitoring parameters represent metrics that are tracked to check the level of compliance with the agreed Service Level Agreements (SLA).

The SLA represents relationship between a service provider and a service customer to ensure and maintain the level of Quality of Services (QoS) to an acceptable level. It needs monitoring, managing and reporting of delivered vs contractual QoS. It guarantees also the quality of the Network Services (NS) to maximize/optimize the ability of the services.

As a general consideration, these parameters can be used for specifying different deployment flavours of a specific service and/or to indicate different levels of service availability. Examples of these parameters are calls per second, maximum number of subscribers, number of rules, flow per second, etc.

2.3.3. Monitoring as-a-service

The 5GCity monitoring systems provide the capability for monitoring both network and cloud infrastructural elements and the related services. Considering the nature of 5GCity heterogeneous and distributed infrastructures and the presence of virtualization components like VMs, a full end-to-end view of both infrastructure and services is required.

To achieve this result, we must have a perspective that the end-to-end route should not be seen as a set of blocks to be optimized locally, but parameter measurements must be analysed from a system perspective to optimize the end-to-end route as a whole. This means the system shall be able to instrument and monitor the different devices composing the overall infrastructure and provide a unique and simple-to-access view of the system that can be exposed to both dashboards and analytical techniques. To this aim, the monitoring module collects and provides all the information needed by the others modules included in the 5GCity platform, e.g. SLA Manager, Slice Manager, in a “monitoring as a service” model.

Each module manages the requested parameters for the specific needs, considering an end-to-end model. Examples of possible application are:

- SLA assurance
- QoS monitoring
- Resource utilization (real-time)
- Resource optimization
- Application health

- System (or sub-systems) health
- Root cause analysis
- Dashboard
- Data analytics

2.3.4. Monitoring module implementation

For the context of 5GCity, Prometheus [16] can be selected as third party monitoring tool to fulfil the monitoring system requirements.

Prometheus is a white box monitoring and alerting system that is designed for large and scalable environments that includes built-in and active scraping, storing, querying, graphing, and alerting based on time series data. It has knowledge about what the world should look like, and actively tries to find faults. Prometheus covers both the domains of instrumentation (using so called “exporters” to instrument platform or applications) and monitoring (providing mechanisms for the data collection, alerting, etc.).

Its main characteristics are:

- A multi-dimensional data model;
- Operational simplicity: easiness to set up monitoring anywhere;
- Scalable and decentralized;
- A powerful query language that uses the data model for meaningful alerting and visualisation.

Figure 24 depicts the internal architecture of the Prometheus system.

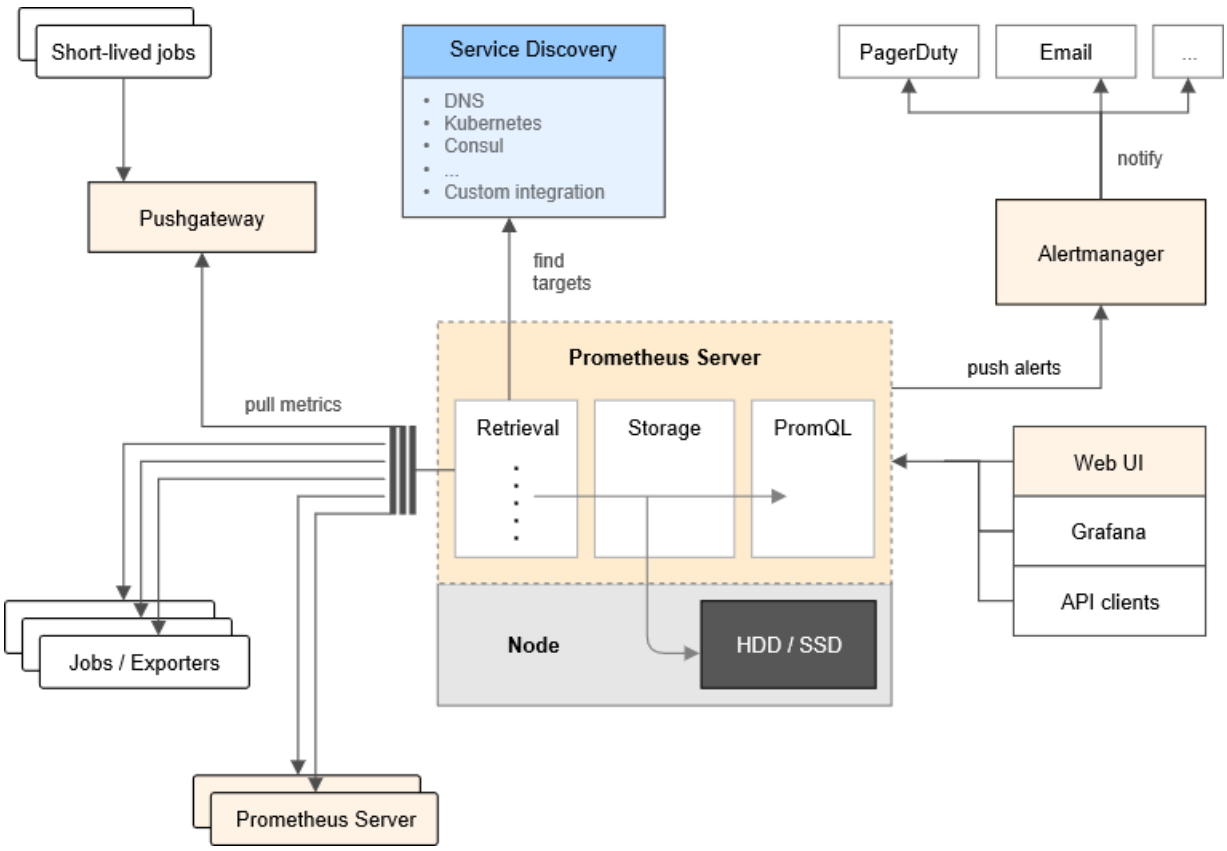

Figure 24. Prometheus internal architecture

Prometheus servers scrape (pull) metrics from instrumented jobs. Additionally, it also supports ephemeral and batch job using "push" method by means of an intermediate service called PushGateway. There is no distributed storage (metrics are stored locally). They can run rules over this data and generate new time series, or trigger alerts. Servers also provide an API to query the data. Metric labelling feature enables easy filtering, grouping, and matching via the query language.

It is able to expose the internal state of running applications giving possibility to send alerts and act upon specific events.

Metrics can be defined by the following metrics type:

- A counter is a metric which is a numerical value that is only incremented, never decremented. Examples include the total amount of requests served, how many exceptions that occur, etc.
- A gauge is an instantaneous metric value that is created via incrementing, decrementing or accumulation. An example could be memory usage, CPU usage, amount of threads, or perhaps a temperature.
- A histogram is a metric that samples observations. A typical use case for a histogram is the measuring of response times.
- A summary is similar to a histogram, but it also calculates configurable quantiles.

Not everything can be instrumented. Third-party tools that do not support Prometheus metrics natively can be monitored with exporters. Exporters can collect statistics and existing metrics, and convert them to Prometheus metrics. An exporter, just like an instrumented service, exposes these metrics through an endpoint, and can be scraped by Prometheus. Prometheus has large number of exporters that export metrics from several systems.

2.4. Security

Access control is essential to guarantee that third parties do not compromise the correct functioning of the system or adulterate sensitive information. Being able to unequivocally identify who is using the system, to know the allowed operations that can be performed by the current user and, what was already done by it, it is essential to build a safe and reliable system. In this context, a three-tier process composed by Authentication, Authorization and Accounting (AAA) must be used.

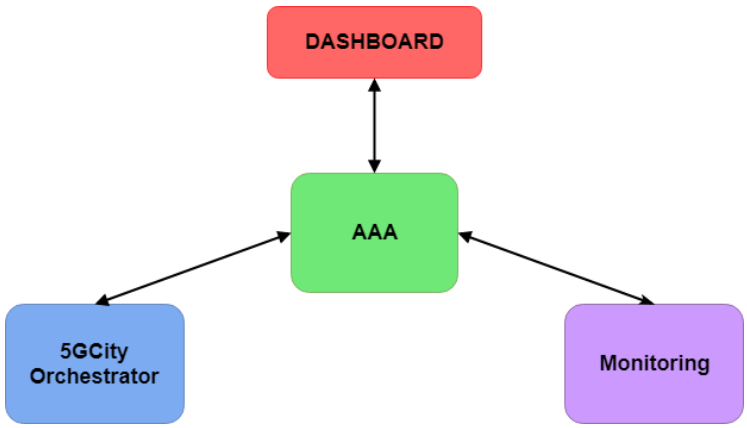

Figure 25. Components that interact with the AAA system

The AAA system needs to be integrated with several 5GCity platform components such as the Dashboard, the 5GCity Orchestrator, and the Monitoring (see Figure 25) in order to secure the interactions between them and the users, but also between the 5GCity Platform components themselves. This security system will mainly support the following operations:

1. Dashboard -> AAA: Registration and login for different users to the 5GCity platform through the dashboard and get credentials for all further actions that will be performed or triggered via the Dashboard.
2. 5GCity Orchestrator -> AAA: Grant access to specific components in the orchestration layer (e.g. Slice Manager). For example, before providing information about a particular slice, the slice manager will check it the requesting user is allowed to access the particular slice. Similarly, for instantiating a service on a particular slice, similar controls will be performed.
3. Monitoring -> AAA: The main goal is to ensure that only authenticated and authorized users have access to monitoring data. Furthermore, in multi-tenant environments, such as 5GCity, it is essential that a slice user has access to its monitoring data only.

Figure 26 illustrates the internals of the AAA process.

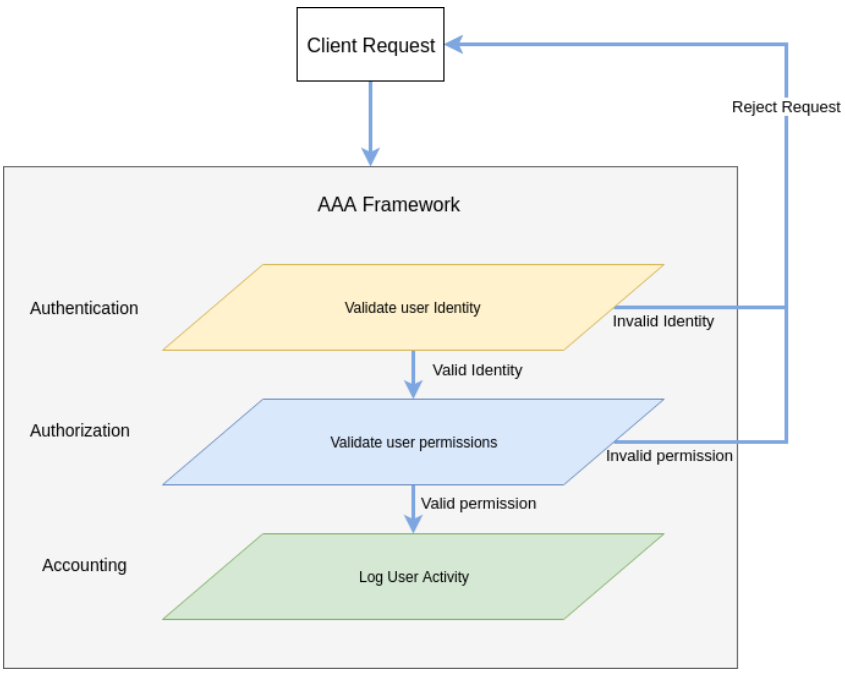

Figure 26. AAA framework approach

Authentication is the first step and aims to identify a given user, *i.e.*, to corroborate if the user is who claims to be usually achieved through a password-based authentication. When registering in the system the user is asked to provide a known unique identifier, usually a username that must be unique and, a password that must be kept a secret. Sometimes, if the system contains sensitive information, the authentication process can be enhanced with other information, *i.e.*, two factor authentication which consists in besides the username and password an extra information that only the user possesses, *e.g.*, a token sent to a mobile phone.

Authorization is the mechanism that asserts what a user can or can't perform within the system, occurring after the authentication. After a registration process, each user has assigned a group of tasks that can perform within the system or a list of resources it can operate on. Authorization is the means to audit these operations guaranteeing a user don't perform tasks it is not allowed to do. The way the accesses are

controlled are, usually, following a Role Based Access Control (RBAC), in which there are a set of roles for a group of tasks being the user only able to perform a task if it has the specified role.

Accounting is the final step within an AAA framework. In order to understand what a given user is doing within a system, *e.g.*, the amount of resources consumed or the amount of data a user sent or receive, the accounting helps to comprehend how the resources are being consumed by a user. By logging all system's activity it's possible to statistically understand the user activity for a wide range of purposes from billing to resource utilization.

Within this project the goal is to have a centralized AAA server that allows all these features, authentication, authorization and accounting in a consolidated and centralized framework capable of handling multiple requests from multiple clients. Third party applications need to interact with the AAA server on behalf of end users performing any action, since every request needs to be authenticated, authorized and accounted.

5GCity will be composed by multiple services providing parallel or complementary features however relying in the same AAA framework to control the access to its features in a controlled and centralized way. To enter into the 5GCIty ecosystem, users will create an account shared across all applications/services residing in the AAA framework. Once a user performs a login operation the request is redirected to the AAA's authentication mechanism which is responsible to validate the provided credentials identifying the user. If the credentials are wrong, the request is rejected, if the credentials are right the authentication returns a token that must be used in other requests, Figure 27. This token is used to identify the user and the current session within the AAA framework avoiding sending the username and password in every request, Figure 28.

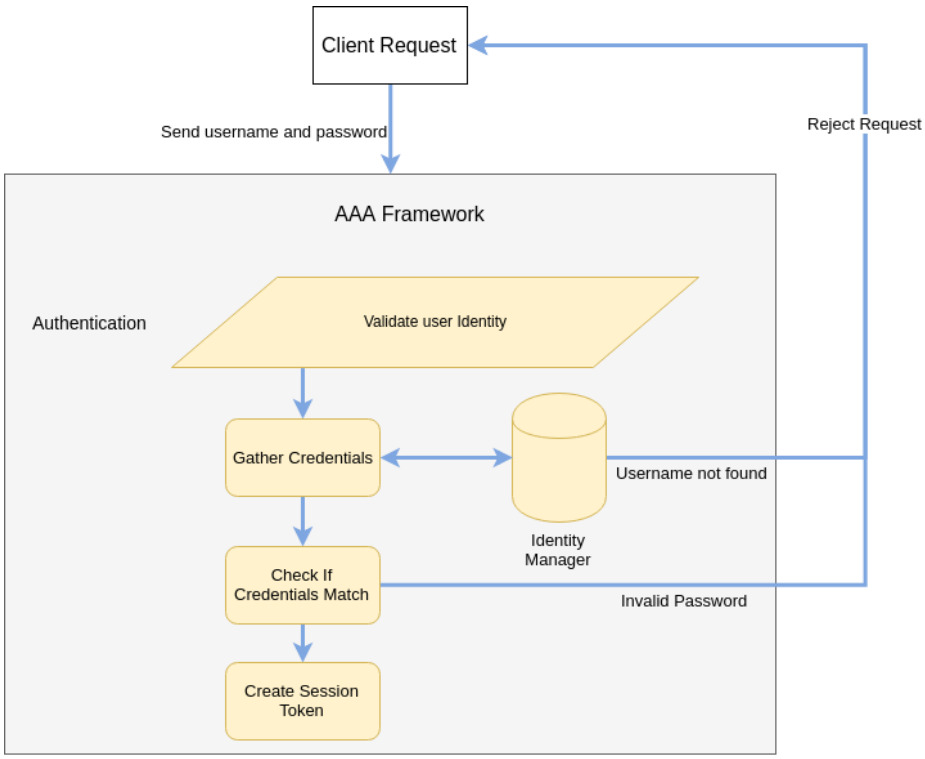

Figure 27. Create session

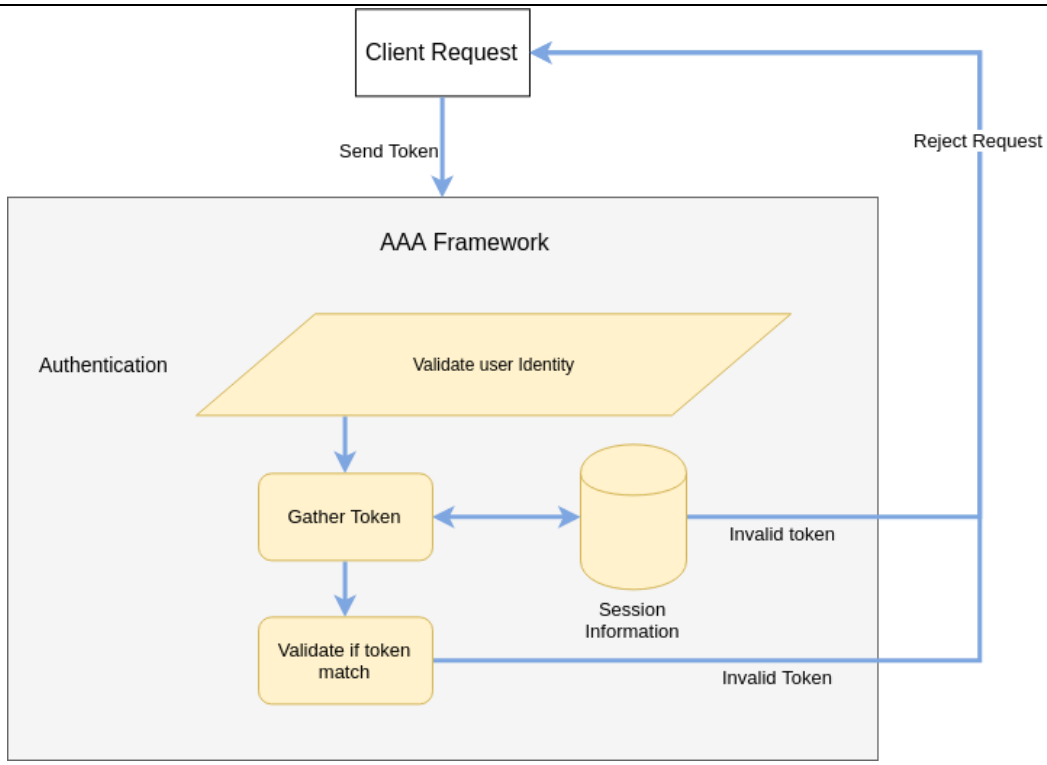

Figure 28. Token-based authentication

Once a user is correctly identified the authorization takes place, *i.e.*, to know if the user has permissions to execute the request. The authorization mechanism works based on a set of centralized rules that dictate what users with a given role can or cannot perform within the system. Once a request arrives to the authorization, the mechanism will search within its rules for the action the user is trying to do, then it will validate if the user has the right role to perform it, Figure 29. When related to resources, the rules can be enhanced with ownership information, *e.g.*, only the user that created the resource can delete it. In order to centralize the process, all 5GCity services using this AAA framework will submit the authorization rules to the AAA following a common format that allows the framework to understand and enforce the rules.

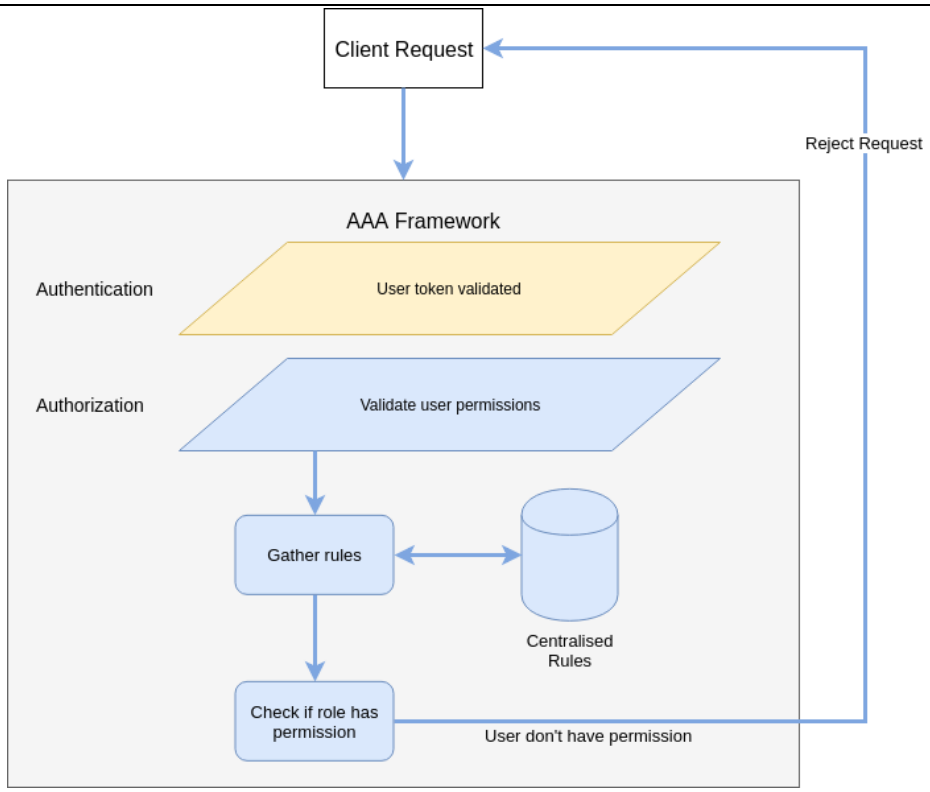

Figure 29. Authorization process

The accounting is the final step of the AAA framework, despite centralized the approach followed on the accounting is slightly different, since the accounting's main goal is to register all user activity for billing (not investigated in 5GCity) or resource control. When a request arrives to the framework it's possible to log what the user accessed, however it's not possible to control how the resources were used, being subordinated to each application/service approach. To address this limitation, besides logging the user's activity each application can notify the AAA framework regarding resource usage and other information. Figure 30 highlights the high-level components need to accomplish this task.

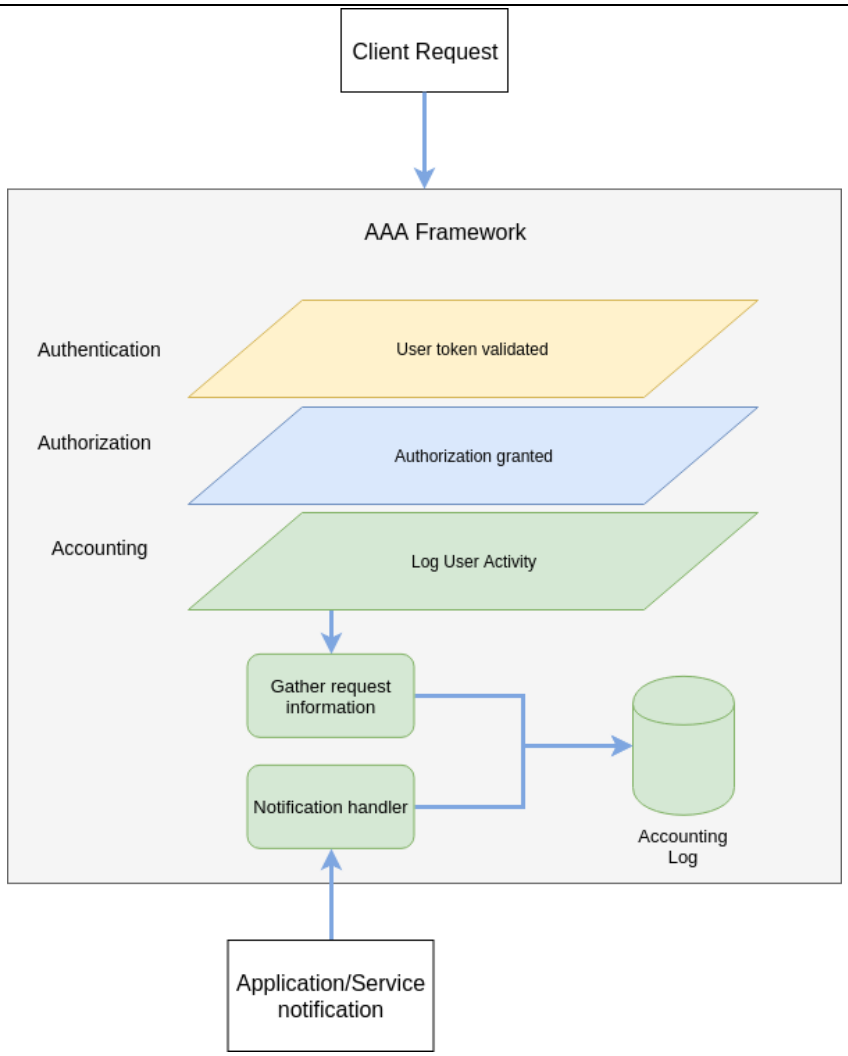

Figure 30. Accounting Process

3. E2E Service Modelling and SDK Toolkit design

Lifecycle management of network services covers a pivotal role within 5G framework. In this section we propose the 5GCity approach which cover the initial phase of a network service lifecycle, namely the design phase, which produces a service template ready to be deployed and operated by a MANO stack over an NFV-enabled architecture.

A proper design of network services is a critical process, which generally should take into account a wide range of aspects going from hardware specific requirements to user SLAs. In this section we propose the following main topics:

- A service modelling strategy which abstracts virtual resources with the main aim to offer to the 5GCity vertical user a simplified view on the infrastructure
- An SDK Toolkit to properly design, craft, and publish end-to-end network services which encompass all three layers defined for 5GCity architecture [1]

3.1. 5GCity service modelling and composition

5GCity Service modelling strategy revolves around the following considerations:

- **Flow based programming strategy**

In computer programming, Flow-Based Programming (FBP) [17] is a programming paradigm that defines applications as networks of "black box" tasks, which exchange data across predefined connections by message passing, where the connections are specified externally to the processes. These black box processes can be reconnected endlessly to form different applications without having to be changed internally. FBP is thus naturally component-oriented.

According to the ETSI NFV definition, a Network Service (NS) is composed by an ordered set multiple Virtual Network Functions (VNF) and can be thought as a pipe, which steer the traffic flows across subsequent stages of data manipulation (Forwarding graph).

According to the given definition, it is clear that the design of a network service can be organized according the rules of flow-based programming, which allows the breakdown of a complex network service into a set of subsequent atomic functions. This approach invokes a clear shift from the concept of a monolithic network service towards a configurable and programmable network service, where proper focus is put on the high degree of re-usability of the single atomic functions of which an overall service is composed. This strategy also led to new business scenarios, where we can envision 5G vertical which is composing its own network services on the basis of atomic functions provided by third-party.

- **Additional Level of Abstraction**

In 5GCity, we look at the municipality as an infrastructure owner acting as Neutral Host who can rent pools of virtual resources (in the form of end-to-end slices) to several customers acting as (“virtual”) Network Operators and Content/Service Providers (e.g. for Media production, broadcast & distribution, connectivity, security). Each Slice User is able to cast its own network services, has its own requirements in terms of services and thus should be able to design network services tailored to the needs of its own business. It is envisioned that the Slice User is enabled to design network services without being aware of full low-level details of the underlying infrastructure. In other words, the infrastructure owner must be able to hide the complexity of the physical infrastructure and expose to its customers only a partial and abstracted view of the resources, according to a model which allows the Slice User to administer the pool of virtual resources it

has rented from infrastructure owner. This model is realized by equipping the 5GCity SDK with an abstraction layer, which is one of the main innovations introduced by 5GCity and offers major advantages in comparison with the current state of the art. Existing SDK toolkits for 5G NFV-enabled infrastructures, e.g. from other EU (European Union) projects like SONATA, Superfluidity, Charisma or embedded in OSM solutions like OSM or Openstack Tacker, still require the user to have a full understanding of MANO information models and awareness of the MANO stack details. Contrary, the 5GCity SDK toolkit adopts a different perspective and allows the user to play with more abstract and business-oriented concepts without caring about infrastructure details, MANO stack procedures, or specific information models.

This overall and two-fold strategy is well depicted in Figure 31 where we can observe the different levels of abstraction enabled by the 5GCity architecture.

- At the upper level (SDK toolkit level), the 5G-vertical user is not aware of low-level details of physical infrastructure and is not aware of MANO NFV information model but is offered a simplified approach to service design and composition. The 5G-vertical user is able, by means of a Graphical tool, to arrange a set of given functions (in this example a Virtual media server, A virtual Firewall and a virtual cache) in an ordered end-to-end service. We also observe that what 5G-vertical user at 5GCity SDK level perceive as single atomic function, can be translated at 5GCity orchestrator level as multiple VNF items each one of them composed by one or more Virtualisation Deployment Unit (VDU).
- At the intermediate level, the 5GCity orchestrator translates the information provided by 5GCity SDK toolkit in an ETSI NFV-compliant model.
- At the lower level (5GCity infrastructure) we see a further translation where the ETSI NFV items are casted over physical infrastructure, which in the 5GCity architecture is divided into three tiers of computing pools of resources (core, edge, radio)

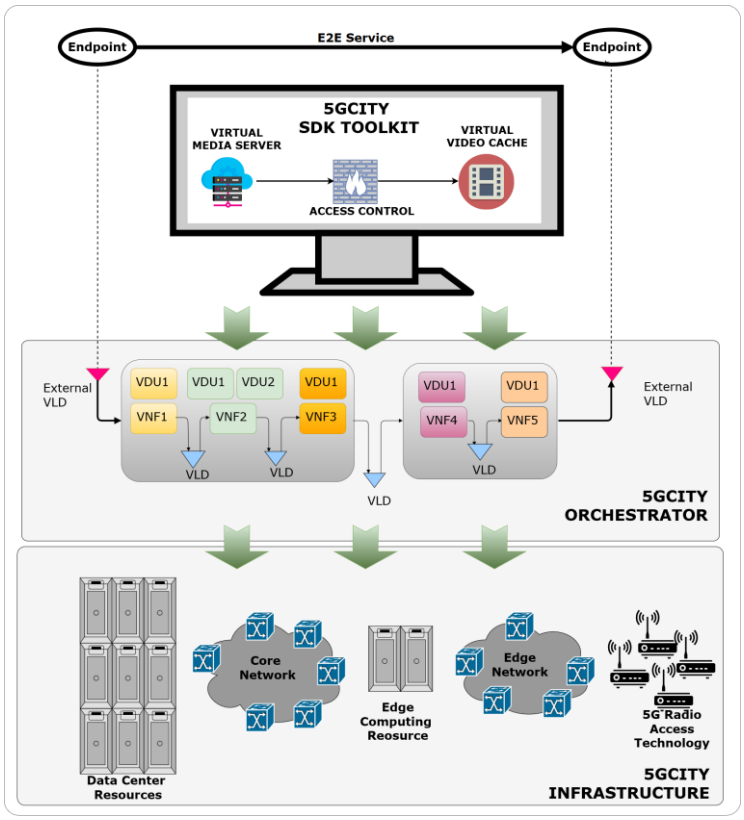

Figure 31. 5GCity with a high level of abstraction.

The 5GCity service modelling strategy has led to the definition of an information model which embodies the 5G vertical user abstracted view of the overall infrastructure. This information model has to be designed as a simplified version of the full ETSI NFV standard Network Service information model.

The different elements composing the information model, their cardinality, and their meaning are described in Table 8, Table 9 and Table 10.

Item	Qualifier	Cardinality	Description
Name	Mandatory	1	User-friendly Name of the Network Service
ID	M	1	UUID identification of the Network Service item
Function	M	1 .. n	Atomic functions which composes the Networks Service. Description of Function on Table 9
Monitoring_Parameters	O	0 .. n	Ordered list of parameters to be monitored by underlying MANO infrastructure. Defined on a per link basis. Each parameter monitoring will have a threshold.
Topology	M	1 .. n	Definition of the topology of the Network service created. Description of the topology on Table 10

Table 8. 5GCity SDK simplified Network Service information model

Item	Qualifier	Cardinality	Description
ID	M	1	UUID identification of the function
Flavour_id	M	1	Deployment flavour of the Atomic Function composing the Network service. It consists in of the virtual resources specification of the VM/container which is used to deploy the actual function.

Table 9. 5GCity definition of atomic function deployment flavour

Item	Qualifier	Cardinality	Description
ID	M	1	Identification of the Topology
Link_ID	M	1	Identification of the Link
Endpoint	M	1 .. n	Identification of the endpoint
SLA	O	0 .. 1	Link level SLA parameters

Table 10. 5GCity definition of Topology

Some notes on the presented information models are provided in the following:

- **flavour_id** represents a combination of three main parameters (vCPU, vRAM, vHDD) which will be later used in MANO environment during the deployment of the Networks Service. Those parameters represent the basic knowledge that infrastructure needs to have to properly build Virtual Machines (or containers) which contains the VNF we are actually considering. 5GCity orchestrator, at deployment time of certain network service, will check if the flavour_id is defined in the VIM infrastructure and in negative case, 5GCity orchestrator will command the VIM its creation.
- **SLA parameters** identified so far are expressed in terms of bandwidth/delay at Link level, to maintain coherency with the ETSI template for NS [14], which assumes that SLA refers to Virtual Link items.
- **Monitoring_Parameters** (consists of an array of parameters and thresholds which provides indication of what needs to be monitored by the MANO infrastructure for the involved service and for which is the threshold, which triggers some actions towards BSS/OSS (alarm/notification) or

towards MANO stack (Network Service scaling). The most common parameters (according to [9]) are the following (and their meaning is self-explanatory):
    - AVERAGE_MEMORY_UTILIZATION
    - DISK_READ_OPS
    - DISK_WRITE_OPS
    - DISK_READ_BYTES
    - DISK_WRITE_BYTES
    - PACKETS_DROPPED
    - PACKETS_RECEIVED
    - PACKETS_SENT
    - CPU_UTILIZATION
- **Topology** (in analogy with the equivalent ETSI NFV VNFFG item) is described as a set of links, each one characterized by a link_id, SLA parameters and the function_id which originates and terminated the link itself. The implicit assumption that network path is symmetric has been done. The detailed structure of the Topology object in the information model is yet to be defined.

This simplified information model will be parsed by a functional block inside the SDK toolkit that will transform the simplified information model into a full ETSI NFV compliant information model.

3.2. SDK Toolkit for Service Programming

SDK toolkit is a self-contained, stand-alone software platform which provides a set of services to support the design of network services ready for deployment upon a 5GCity infrastructure.

SDK toolkit is able to selectively offer different functionalities depending on the user/role which is exploiting its functionalities. This design ensures a high level of configurability of the SDK, which is able to be deployed in a wide range of scenarios, by a wide range of users, and agnostic with regard to the underlying infrastructure.

5GCity SDK toolkit can be used in several deployment scenarios, namely it can be deployed by:

- Slice Users, e.g. in order to develop the NSs and VNFs that they will deploy on their own slices
- Neutral hosts, e.g. in order to develop elements required by the 5GCity platform in order to serve the needs of the Slice Users
- Service Developers (belonging to any third party), e.g. in order to develop and register VNFs that can be “sold” to Slice Users

The relationships between SDK toolkit and the rest of the platform are described in Deliverable D2.2 [1]. Proper design of the SDK toolkit allows for a coherent integration with the underlying architecture, in a transparent way with regard to the different deployment scenarios.

3.2.1. Requirements

SDK Toolkit requirements have been derived based on the following:

- High-level requirements gathered from the project proposal
- State of the art of similar platforms as described in relevant 5G projects (SONATA, 5GTANGO, SESAME, Superfluidity)

The list of requirements identified so far is described in Table 11.

Area	Req-id	Requirement name	TYPE	Description
user management	REQ-USER-1	Multiple user handling	MANDATORY	SDK must be able to handle multiple users.
user management	REQ-USER-2	Workspace definition	MANDATORY	Each user has to be provided with suitable workspace virtually separated by other users.
user management	REQ-USER-3	Multiple Roles handling	MANDATORY	SDK must be able to handle different roles. The following roles are foreseen: Admin, Vertical, and Dev-ops
user management	REQ-USER-4	Admin Role definition	MANDATORY	Admin Role is in charge of maintenance, configuration of the interface with full access to each for the components composing the SDK
user management	REQ-USER-5	Vertical role definition	MANDATORY	Vertical Role is in charge of SDK utilization, with limited access to its toll. Vertical user is exposed a simplified view of the virtual resources. Vertical role is not aware of NFV MANO information model and related low-level details. Vertical role is allowed to design network services according a simplified information model which is presented in sec 2.1.
user management	REQ-USER-6	Dev-ops role definition	OPTIONAL	Dev-ops role is in charge of the SDK utilization, with full access to GUI-CLI tools.
Functional	REQ-FUNC-1	Composer	MANDATORY	SDK must be equipped with composer tool, which Ensure for the role=Vertical a basic set of capabilities is unlocked.
Functional	REQ-FUNC-2	Composer	MANDATORY	Composer tool provides a basic set of capabilities which allow user with Role=vertical to • Retrieve a set of given atomic functions • Create/Edit/delete Network service composed as an ordered set of functions.
Functional	REQ-FUNC-2	Composer	MANDATORY	Composer tool contains an adaptation module which maps the 5GCity SDK vertical information model to a full ETSI NFV information model.
Functional	REQ-FUNC-2	Composer	MANDATORY	Composer tool contains a validation module which operates formal validation of Network Service Templates designed by the user against standard information model as described in

				ETSI GS NFV-SOL 004 V2.3.1 and Draft ETSI GS NFV-SOL 001 V0.3.0 (2017-11)
Functional	REQ-FUNC-1	Editor tool	OPTIONAL	SDK must be equipped with editor module which unlocks full capability of SDK toolkit. Editor tool enable user with dev-ops role to edit-modify delete VNF/NS stored in the private catalogue.
Functional	REQ-FUNC-6	Delivery tool	MANDATORY	SDK should be equipped with a module to promote-demote items for private catalogue to public catalogue.
Functional	REQ-FUNC-7	Private Catalogue	MANDATORY	SDK must be equipped with Private Catalogue where user stores crafted items.
Functional	REQ-FUNC-9	Emulation tool	OPTIONAL	SDK must be equipped with Emulation tool which is able to emulate the deployment of user created services over emulated MANO stack.
Miscellaneous	REQ-MISC-1	deployment	OPTIONAL	SDK should be provided in the form of script-installer which take care of the installation and configuration of the framework
Miscellaneous	REQ-MISC-1	deployment	OPTIONAL	SDK should be self-contained cloud-ready VM-Container

Table 11. Requirements for SDK Toolkit

3.2.2. Architecture

In the following section, the description of the design for the 5GCity SDK toolkit is provided. The selected architecture has been designed in order to fully cover the high-level requirements as expressed in the previous section, by taking care of the following main functionalities:

- **Graphical User Interface** represents the entry point for the platform and allows users to interact with services exposed by SDK toolkit. Operation performed by the user at GUI level are translated in API calls towards Composer and Editor blocks
- **Composer** is a set of tools which allows the user (Role=Vertical) to manage Network Service items by providing the following capabilities:
  - Intercept commands (Create, modify, and delete operations) issued by user via GUI
  - Interact with local database with the main purpose of storing/retrieving items
  - Adapt the 5GCity SDK toolkit information model to fully ETSI NFV information model
  - Validate the network service items created by user
- **Editor** (OPTIONAL, described here only for reference) is a set of tools which allows the user (Role=dev-ops) to manage Network Service items by providing the following capabilities:
  - Intercept commands (Create, modify, and delete operations) issued by user via GUI
  - Interact with local database with the main purpose of storing/retrieving items
  - Validate the network service items created by user
- **Policy Engine** is the function block which implements RBAC access to resources, by checking the matching between user/role which issue the request and the requested resource

- **Private App service and Application Catalogue** is a collection of services which provides the following capabilities:
    - A dispatcher engine which takes care of adapting the API calls sent-received on North Bound interfaces in order to be served by local database and MANO plugin
    - A local database where the NFV items packages (NS, VNF) are store according a full ETSI information model
    - A MANO plugin which is able to translate a generic NS descriptor to a NS descriptor which is related to a specific MANO stack
- **Emulation Toolkit** (OPTIONAL, described here only for reference) provides an emulated full stack MANO framework where the user is allowed to deploy the Network service designed by means of the 5GCITY SDK Toolkit.

The 5GCity SDK toolkit has been designed in order to implement RBAC access to the resources. A clear definition of user and roles is provided in the following table, together with matrix which states correlated the roles with the operations they are allowed to perform on the resources.

Role Type	**Role Description**	**Resources accessed and allowed operations**
Role=Admin	It is in charge of the configuration and maintenance of the SDK framework with full access (super-user) to each of the components composing the SDK	• Life Cycle management of the platform.
Role=Vertical	It has access to a basic set of SDK functionalities, mainly consisting in the editing of a service item as an ordered set of given (atomic) functions. The Vertical role is exposed a simplified-abstracted view of 5GCITY resources.	• Full R/W access to NSd • ReadOnly access to VNF packages
Role=Developer	It has access to the full capabilities of 5GCITY-SDK, being able to create/modify VNF and NS templates	• Full R/W access to NSd • Full R/W access to VNF packages

Table 12. 5GCity SDK roles definition and access to catalogue resources

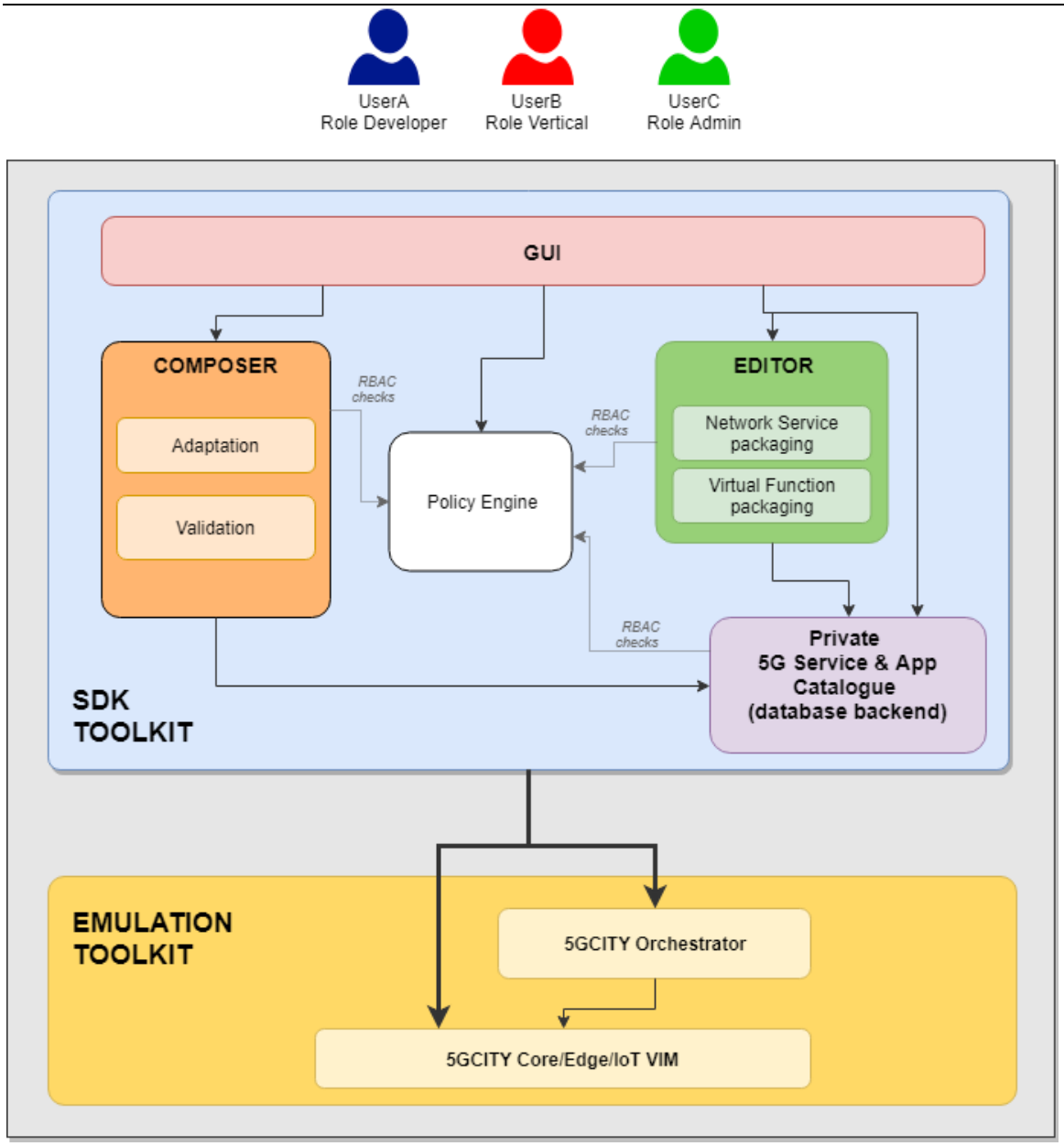

Figure 32. SDK Toolkit high-level architecture.

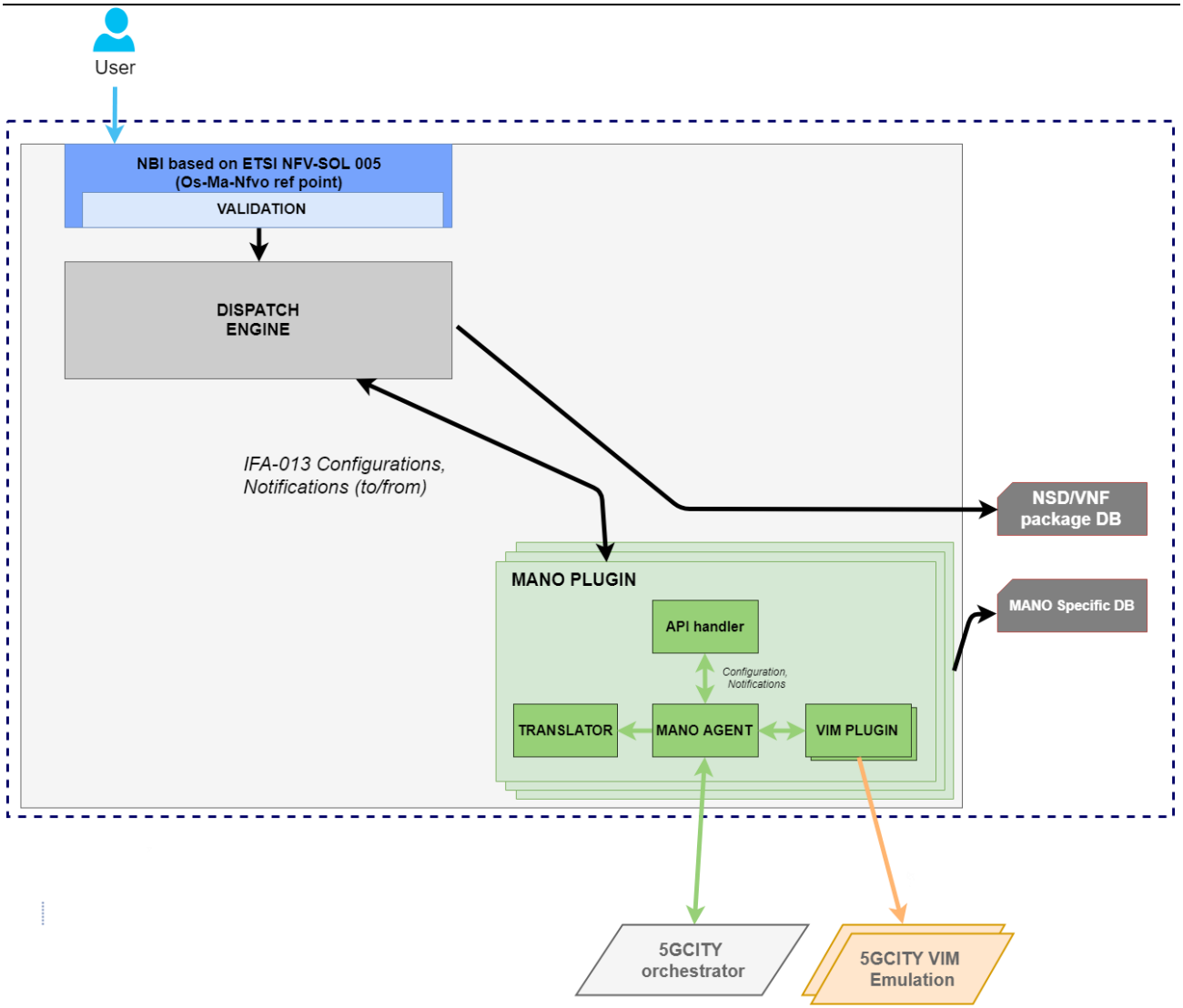

Figure 33. Private 5G Service & Application Catalogue.

Private Catalogue is described in Figure 33 and it is composed by the following main elements:

- Dispatch Engine which is in charge of delivering messages among catalogue main components
- Database back-end where templates are stored
- MANO plugin which is in charge to translate a deployment template according the generic ETSI NFV information model, to an information model which is specific to the MANO platform deployed.

3.2.3. Information Models

ETSI NFV standard provides a clear definition of the information model used represent items, relationships, constraints and operations and a modelling language used for the for deploying of applications and services in virtual, physical, or hybrid networks. TOSCA, a data modelling standardization effort led by Oasis for cloud orchestration environments and applications, has been selected as a modelling language for the ETSI NFV deployment templates [ETSI GS NFV SOL04]. The TOSCA NFV profile [tosca-nfv-v1.0-csd04] specifies an NFV specific data model using TOSCA language.

The deployment and operational behaviour requirements of each Network Service in NFV is captured in a deployment template and stored during the Network Service on-boarding process in a catalogue, for future selection for instantiation. This profile using TOSCA as the deployment template in NFV and defines the NFV

specific types to fulfil the NFV requirements. This profile also gives the general rules when TOSCA used as the deployment template in NFV.

ETSI NFV information model is composed by four information elements defined apart from the top-level Network Service (NS) information element:

- Virtualized Network Function (VNF) information element
- Physical Network Function (PNF) information element
- Virtual Link (VL) information element
- VNF Forwarding Graph (VNFFG) information element

Those information models should be wrapped in suitable deployment template, which provides all the necessary information details for the deployment upon a full NFV MANO infrastructure, namely:

- A VNF Descriptor (VNFD) is a deployment template which describes a VNF in terms of its deployment and operational behaviour requirements.
- A VNF Forwarding Graph Descriptor (VNFFGD) is a deployment template which describes a topology of the Network Service or a portion of the Network Service, by referencing VNFs and PNFs and Virtual Links that connect them.
- A Virtual Link Descriptor (VLD) is a deployment template which describes the resource requirements that are needed for a link between VNFs, PNFs and endpoints of the Network Service, which could be met by various link options that are available in the NFVI.
- A Physical Network Function Descriptor (PNFD) describes the connectivity, Interface and KPI (Key Performance Indicator) requirements of Virtual Links to an attached Physical Network Function.

The main assumption used for TOSCA modelling language within an NFV domain is the following:

- ETSI NFV NSD is modelled as a TOSCA Service template
- ETSI NFV VNFD, VLD, VNFFG, PNFD are modelled as Node template

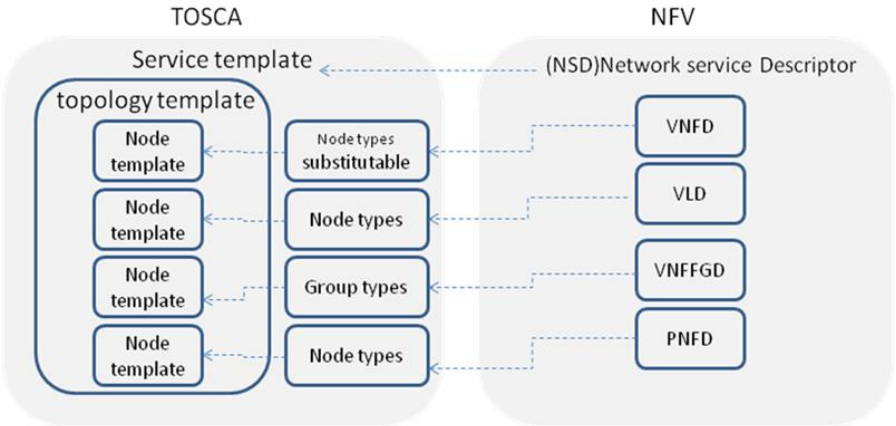

Figure 34. ETSI NFV deployment template described with TOSCA model mapping.

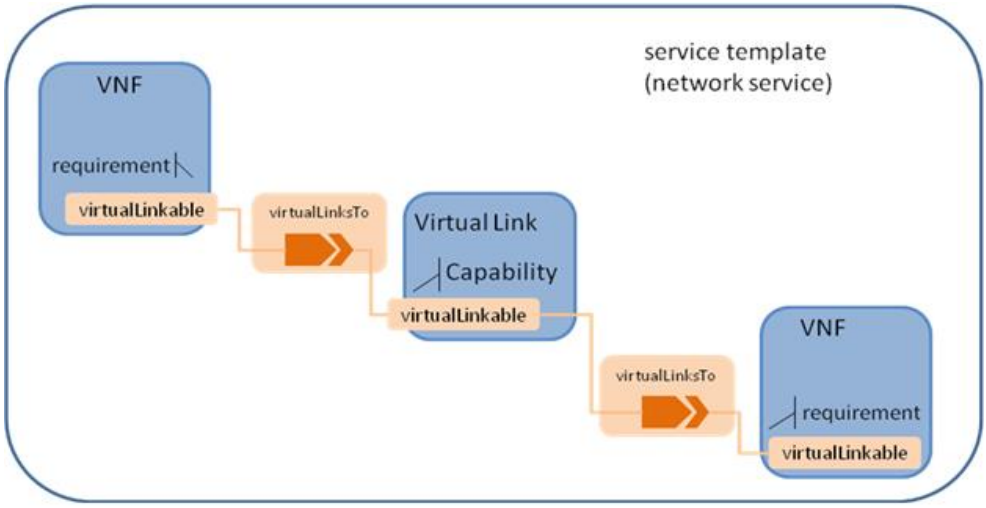

Figure 35. TOSCA deployment template modelling ETSI NFV

Figure 34 and Figure 35 describe an example of TOSCA resource modelling, where VNFD and VLD are represented as Node Templates and a new "virtualLinksTo" relationship type can be defined to connect VNF and VL.

3.2.4. Main interfaces

SDK toolkit is designed with a set of interfaces which allows external communication between SDK toolkit and the 5GCity service & App Public catalogue on one hand and internal interaction among the different entities composing the platform.

SDK Toolkit interfaces as described in Figure 36 are identified as follows:

- SDK.E1 (external interface to Public Catalogue)
- SDK.I1 (internal interface between GUI and Composer)
- SDK.I2 (internal interface between GUI and Policy Engine)
- SDK.I3 (internal interface between Composer and Private Catalogue)
- SDK.I4 (internal interface between GUI and Editor)

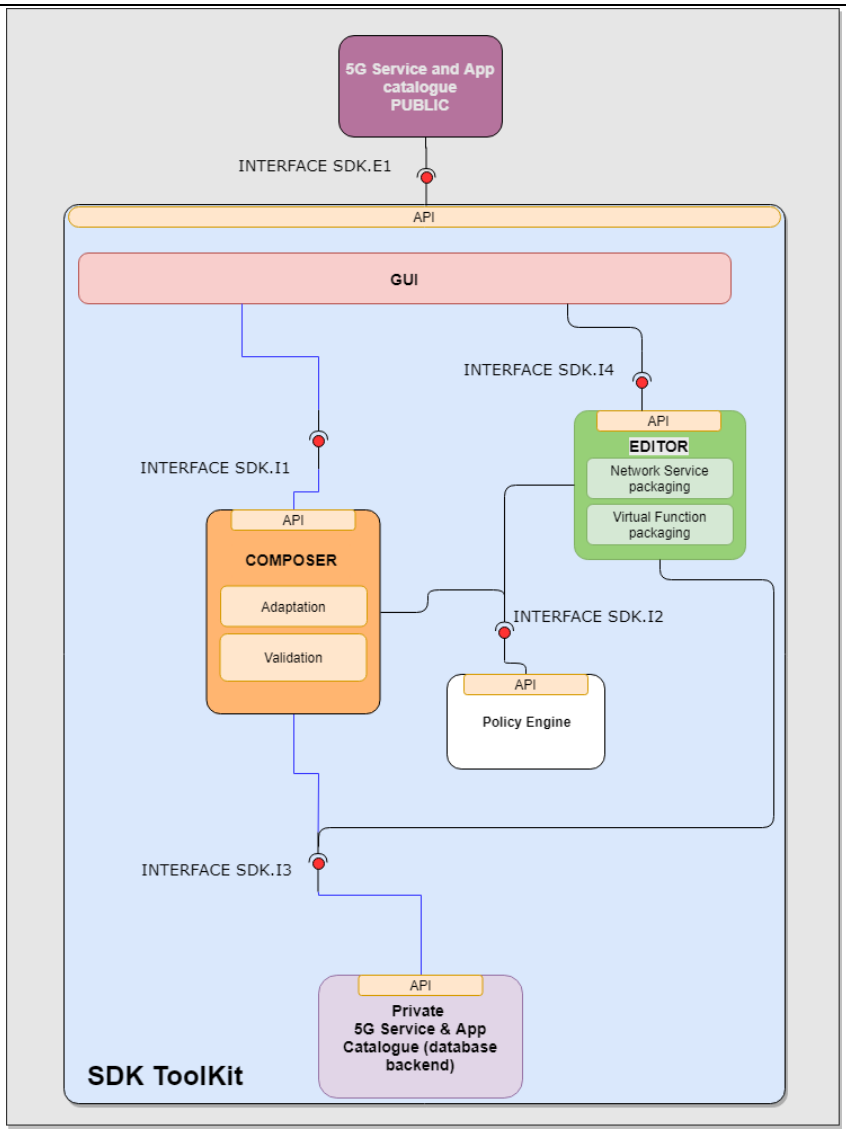

Figure 36. SDK internal Interfaces diagram

3.2.4.1.External Interfaces

5GCity SDK External interface, SDK.E1 allows communication between SDK Toolkit and the Public catalogue platform. SDK can perform the following main operations:

- Onboard NSD package to 5G Service & App Public Catalogue
- Modify/deleted NSD package residing in 5G Service & App Public Catalogue
- Fetch NSD package from 5G Service & App Public Catalogue
- Onboard VNF package to 5G Service & App Public Catalogue
- Modify/deleted VNF package residing in 5G Service & App Public Catalogue
- Fetch VNF package from 5G Service & App Public Catalogue

This interface is modelled according ETSI NFV Os-Ma-Nfvo reference point [18] is exposed by 5G Service & App Public catalogue and is designed according the following requirements which are directly inherited from Os-Ma-Nfvo.NSd interface definition [18]. The numbering has been maintained to stress the close relationship between 5GCITY SDK external interface and the its equivalent ETSI NFV. Original Os-Ma-Nfvo.NSd requirements which are not applicable to 5GCITY scenario have been left blank.

Numbering	Functional requirement description
SDK.E1.Nsd.001	The NSD Management interface produced by the PUBLIC CATALOGUE on the SDK.E1 reference point shall support on-boarding NSD.
SDK.E1.Nsd.002	The NSD Management interface produced by the PUBLIC CATALOGUE on the SDK.E1 reference point shall support disabling an NSD.
SDK.E1.Nsd.003	The NSD Management interface produced by the PUBLIC CATALOGUE on the SDK.E1 reference point shall support enabling an NSD.
SDK.E1.Nsd.004	The NSD Management interface produced by the PUBLIC CATALOGUE on the SDK.E1 reference point shall support updating an NSD..
SDK.E1.Nsd.005	The NSD Management interface produced by the PUBLIC CATALOGUE on the SDK.E1 reference point shall support querying NSDs.
SDK.E1.Nsd.006	The NSD Management interface produced by the PUBLIC CATALOGUE on the SDK.E1 reference point shall support deleting an NSD.
SDK.E1.Nsd.007	The NSD Management interface produced by the PUBLIC CATALOGUE on the SDK.E1 reference point shall support providing notifications about the on-boarding of NSDs.
SDK.E1.Nsd.008	The NSD Management interface produced by the PUBLIC CATALOGUE on the SDK.E1 reference point shall support providing notifications as a result of changes on NSD states.
SDK.E1.Nsd.009	N/A
SDK.E1.Nsd.010	N/A
SDK.E1.Nsd.011	N/A
SDK.E1.Nsd.012	N/A
SDK.E1.Nsd.013	The NSD Management interface produced by the PUBLIC CATALOGUE on the SDK.E1 reference point shall support subscribing to notifications related to NSD management changes.
SDK.E1.Nsd.014	The NSD Management interface produced by the PUBLIC CATALOGUE on the SDK.E1 reference point shall support fetching an NSD.
SDK.E1.Nsd.015	N/A
SDK.E1.Nsd.016	N/A
SDK.E1.Nsd.017	N/A

Table 13. SDK.E1 NS interface requirements

SDK.E1.VnfPkgm.001	The VNF Package Management interface produced by the PUBLIC CATALOGUE on the Os-Ma-nfvo reference point shall support on-boarding a VNF Package.
SDK.E1.VnfPkgm.002	The VNF Package Management interface produced by the PUBLIC CATALOGUE on the Os-Ma-nfvo reference point shall support disabling a VNF Package.
SDK.E1.VnfPkgm.003	The VNF Package Management interface produced by the PUBLIC CATALOGUE on the Os-Ma-nfvo reference point shall support enabling a VNF Package.
SDK.E1.VnfPkgm.004	The VNF Package Management interface produced by the PUBLIC CATALOGUE on the Os-Ma-nfvo reference point shall support querying VNF Package information. See note 1.
SDK.E1.VnfPkgm.005	The VNF Package Management interface produced by the PUBLIC CATALOGUE on the Os-Ma-nfvo reference point shall support deleting a VNF Package.
SDK.E1.VnfPkgm.006	The VNF Package Management interface produced by the PUBLIC CATALOGUE on the Os-Ma-nfvo reference point shall support providing notifications about the on-boarding of VNF Packages.
SDK.E1.VnfPkgm.007	The VNF Package Management interface produced by the PUBLIC CATALOGUE on the Os-Ma-nfvo reference point shall support providing notifications as a result of changes on VNF Package states.
SDK.E1.VnfPkgm.008	The VNF Package Management interface produced by the PUBLIC CATALOGUE on the Os-Ma-nfvo reference point shall support fetching a VNF Package, or selected artifacts contained in a package.
SDK.E1.VnfPkgm.009	N/A
SDK.E1.VnfPkgm.010	The VNF Package Management interface produced by the PUBLIC CATALOGUE on the Os-Ma-nfvo reference point shall support updating VNF Package information. See note 2.

Table 14. SDK.E1 VNF interface requirements

3.2.4.2.Internal Interface SDK.I1

5GCity SDK External interface, SDK.I1 allows communication between GUI front-end and the Composer entity. The interface is produced by the Composer entity and allows the operations described in Table 15 and Table 16 for two different type of resources.

Operation Type	Resource	Parameters	Metadata	Description	Expected Result
GET	/sdki1/function	Function_ID	Authentication Data	retrieve resource identified with Function_ID	All data and metadata related to the Function_ID

Table 15. SDK.I1 interface description for the function resource

Regarding the NetworkService resource, the interface permits all CRUD operation, shown on the following table.

Operation Type	Resource	Parameters	Metadata	Description	Expected Result
GET	/sdki1/ns	NetworkService_ID	Authentication Data	retrieve resource identified with Network_Service_ID	All data and metadata related to the NetworkService_ID
POST	/sdki1/ns	Function_ID_1, VM_Flavour … Function_ID_N, VM_Flavour Topology Monitoring parameters Service SLA parameters	Authentication Data	Creation of NetworkService Element	. A NetworkService_ID, all its data and related metada
PUT	/sdki1/ns	NetworkService_ID Function_ID_1, VM_Flavour … Function_ID_N, VM_Flavour Monitoring parameters Link SLA parameters	Authentication Data	Update of NetworkService Element.	A NetworkService_ID all its data and related metada are retrieved
DELETE	/sdki1/ns	NetworkService_ID	Authentication Data	Deletion of NetworkService Element	Status of the operation (OK, NotOK)

Table 16. SDK.I1 interface description for the NetworkService resource

3.2.4.3.Internal Interface SDK.I2

5GCity SDK internal interface, SDK.I2 allows communication between Composer and Policy Engine, to ensure that Composer access operations to the resources stored in the private catalogue are checked against user and role privileges. The interface is produced by the Policy Engine and allows the operations described in Table 17.

Operation Type	Resource	Parameters	Metadata	Description	Expected Result
GET	/sdki2/token	User, Credentials Resource_ID	N/A	Retrieve privilege to access the Resource element by provided user	Token, token validity

Table 17. SDK.I2 interface description

3.2.4.4.Internal Interface SDK.I3

5GCity SDK internal interface, SDK.I3 allows communication between Composer and Private catalogue. The interface is produced by Private Catalogue and allows the operations as described in Table 18 and Table 19.

Operation Type	Resource	Parameters	Metadata	Description	Expected Result
GET	/sdki3/vnfd	VNFd_id	Authentication Data	retrieve resource identified with VNF_id	All data and metadata related to the VNFd package
POST	/sdki3/vnfd	VNFd_id	Authentication Data	Creation of VNFd package Element	Full VNF package
PUT	/sdki3/vnfd	VNFd_id	Authentication Data	Update of VNFd package Element	Full VNF package
DELETE	/sdki3/vnfd	NSd_id	Authentication Data	Deletion of NetworkService Element	Status of the operation (OK, NotOK)

Table 18. SDK.I3 interface description for VNFD resources

Operation Type	Resource	Parameters	Metadata	Description	Expected Result
GET	/sdki3/ns	NSd_id	Authentication Data	retrieve resource identified with Network_Service_ID	All data and metadata related to the NSd package
POST	/sdki3/ns	NSd Package	Authentication Data	Creation of NetworkService Element	Full NSD package
PUT	/sdki3/ns	NSd Package	Authentication Data	Update of NetworkService Element	Full NSD package
DELETE	/sdki3/ns	NSd_id	Authentication Data	Deletion of NetworkService Element	Status of the operation (OK, NotOK)

Table 19. SDK.I3 interface description for NS resources

3.2.4.5.Internal Interface SDK.I4

5GCity SDK internal interface, SDK.I4 allows communication between GUI and Editor module. The interface is produced by Editor Module and allows the operations as described in Table 20

Operation Type	Resource	Parameters	Metadata	Description	Expected Result
GET	/sdki4/vnfd	VNFd_id	Authentication Data	retrieve resource indentified with VNF_id	All data and metadata related to the VNFd package

Operation Type	Resource	Parameters	Metadata	Description	Expected Result
GET	/sdki4/vnfd	NSd_id	Authentication Data	retrieve resource indentified with Network_Service_ID	All data and metadata related to the NSd package
POST	/sdki4/vnfd	NSd Package	Authentication Data	Creation of NetworkService Element	. Full NSD package
DELETE	/sdki4/vnfd	NSd_id	Authentication Data	Deletion of NetworkService Element	Status of the operation (OK, NotOK)

Table 20. SDK.I4 interface description

3.2.5. Main interactions

In the next sections, sequence diagrams will be provided to show the interaction between the modules composing the SDK in the following cases:

- Login to GUI
- Network Service Creation (Role=vertical)
- Network service Creation (Role=developer)

The sequence diagrams will also provide clear indication regarding how the same operation (Service Creation) involves different modules and result in different workflows according the user role which is triggering the initial request.

Note that in this deliverable only sequence diagram for positive cases will be described. The sequence diagrams which describe the SDK state machine completely will be provided in the next WP4 deliverables.

3.2.5.1.User Login to GUI

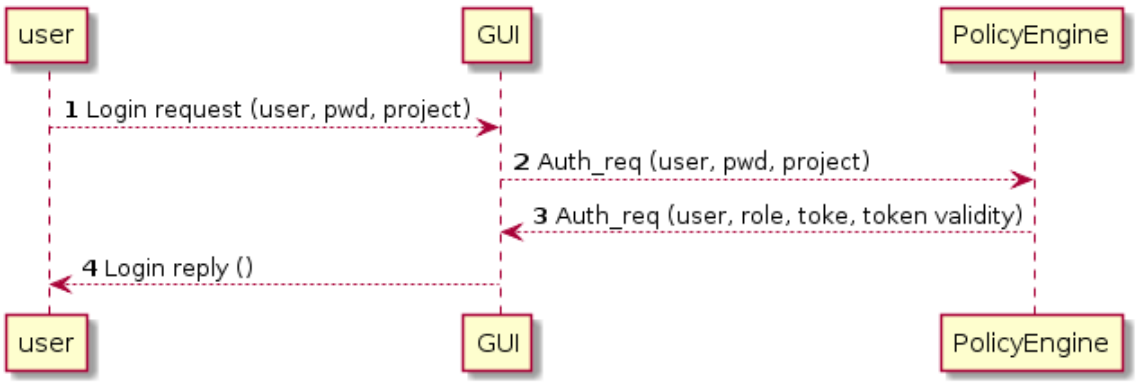

Figure 37. Workflow for user login to SDK GUI.

The sequence diagram which describes the SDK processing a login request done by the user is structured as follows:

1. User initiate a login request, by providing user and password
2. GUI front-end receive the user credentials and send an Auth_req message to the Policy Engine module
3. Policy Engine Module, according to its database where user credentials are stored, provides an Auth_reply message which contains for the current user a software token and a token time frame validity.
4. GUI front-end provides a successful login reply to the user. SDK home page for the current user/role combination is displayed and made accessible to the user.

3.2.5.2.User with role=Vertical, Service creation

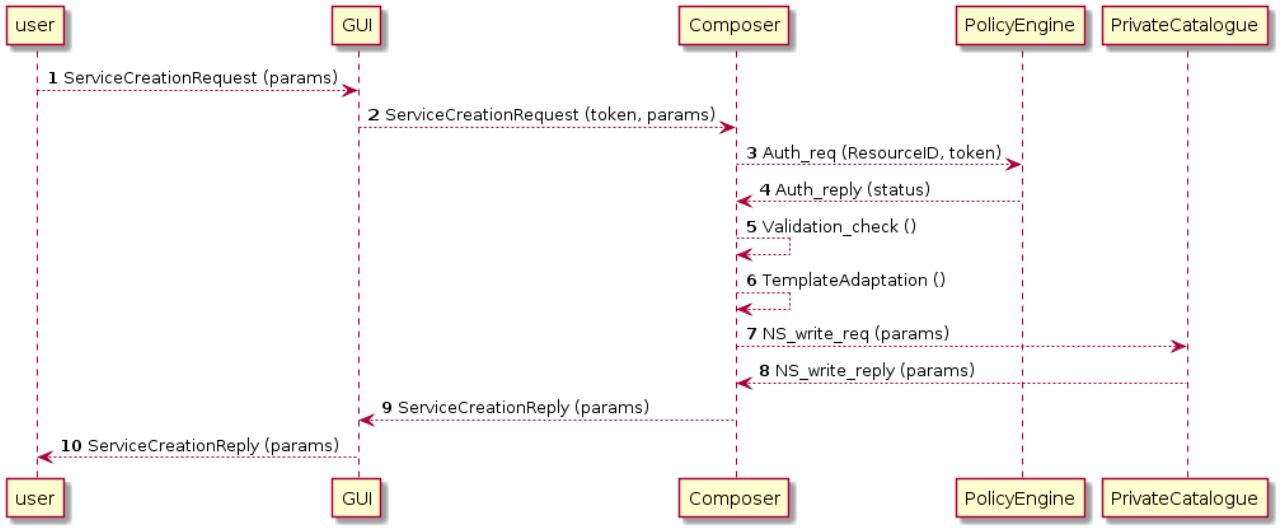

Figure 38. Sequence diagram for Service Creation (role=vertical).

The sequence diagram which describes the SDK processing the request to create a network service done by the user with role=vertical is structured as follows:

1. User logged in to SDK GUI, issue the request for the creation of Network Service, by providing the necessary parameters (as described in sec 3.1)
2. GUI front-end parse the parameters provided by user and send a ServiceCreationRequest to Composer Module. The message also software token that user has previously obtained at login time
3. Composer issues an Auth_req to PolicyEngine module providing software token and identification for resources to be accessed
4. Policy Engine grants access by replying with Auth_reply message
5. Composer performs a validation of the parameters in the ServiceCreationRequest message
6. Composer performs a template adaptation operation by translating SDK simplified information model to a full ETSI NFV compliant information model
7. Composer sends to private Catalogue a NS_write_Req by requesting the store of the newly created Network Service template into the local database

3.2.5.3.User with role=developer, Service creation

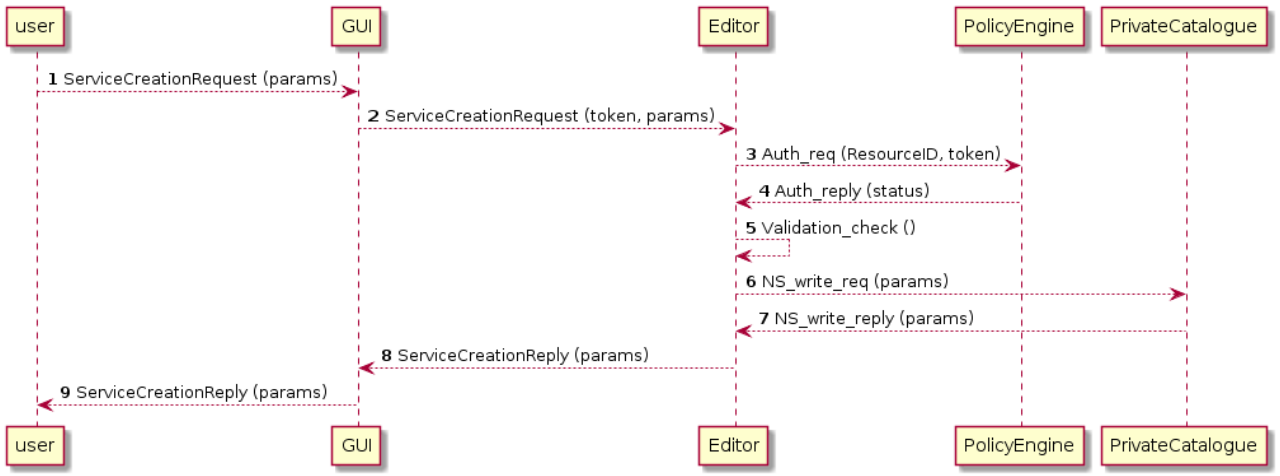

Figure 39. Sequence diagram for Service Creation (role=developer)

The sequence diagram which describes the SDK processing the request to create a network service done by the user with role=developer is structured as follows:

1. User logged in to SDK GUI, issue the request for the creation of Network Service, by providing the necessary parameters (as described in sec 3.1)
2. GUI front-end parse the parameters provided by user and send a ServiceCreationRequest to Editor Module. The message also software token that user has previously obtained at login time.
3. Editor issue a Auth_req to PolicyEngine module providing software token and identification for resources to be accessed
4. Policy Engine grants access by replying with Auth_reply message
5. Editor perform a formal validation of the parameters contained in the ServiceCreationRequest message

Editor sends to private Catalogue an NS_write_Req by requesting the store of the newly created Network Service template into the local database

4. Federated Machine Learning

Recent hardware developments have dramatically increased parallelism of computation by orders of magnitude. The commoditization of hardware such as GPUs, along with their wide deployment not only in data-centers but also in personal computers and mobile devices, means that nowadays it is possible to run large mathematical computations that would have in the past required super computers.

One field that has particularly benefitted from these trends is Machine Learning (ML), which leverages massive parallelism in order to train ML models quickly and cheaply. Within ML, Neural Networks (NNs) have recently seen its popularity increase thanks to its ability to yield excellent results in image recognition, voice recognition and language translation, among others.

By federated machine learning we mean the computation of such NNs in distributed, heterogeneous settings which may include large data-centers with clusters of standard x86 servers, but also edge equipment running on single-board computers all the way down to mobile devices. Although GPUs (Graphics Processing Units) are now standard in such mobile devices, their performance is still worse than GPUs installed in rack-scale computers. Further, mobile devices are constrained in terms of network bandwidth and battery, but have the clear advantage that, if computations are done on the device itself, delay is potentially shorter and network bandwidth consumption is null.

Figure 40 depicts the three-tier structure of federate ML scenarios, with compute power in data-centers, at edge nodes and on mobile device; this precisely maps to 5GCity's three-tier architecture, with mobile devices, edge devices (on lamp posts or city cabinets) and DCs (in municipality buildings). Given this, the aim of this work is to optimize the computation of NNs over such distributed infrastructure (e.g., for the illegal waste dump use case, which relies on NNs to detect the illegal dumping).

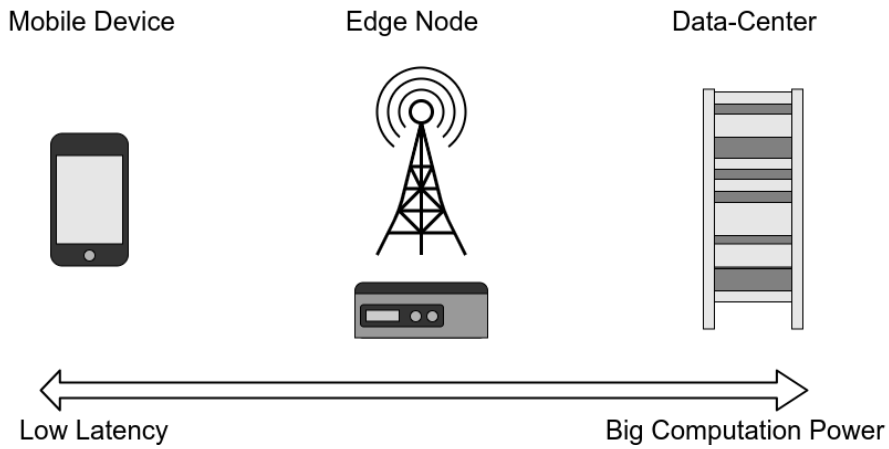

Figure 40. The edge computing architecture.

Compute resources of edge nodes are usually larger than mobile devices, but smaller than machines in data-centers. However, edge nodes are geologically distributed and located near to users' mobile devices so that the mobile devices can leverage bigger computation resources with low latency. In short, the edge computing technology allows mobile devices to extend their available computation resources and increase the number of runnable applications of them.

In our federated machine learning platform, we investigate the numerous trade-offs between running training or prediction NN workloads on these three types of devices, taking into account metrics such as CPU and GPU power, power consumption, delay and network bandwidth, among others. We call our initial framework the FML (Federated Machine Learning) platform, which is able to distribute NN workloads across a set of distributed compute nodes. We show results from initial performance tests to give a taste of FML's potential benefits.

4.1. 5GCity Federated Machine Learning platform

Although in 5GCity configuration parameters of the edge infrastructure (e.g., RAN allocations, edge connectivity flow tables) are in principle set by the WAN Resource Manager and the VIMs based by centralized optimization logic of the 5GCity Orchestrator, in some case fast, local decisions and reactions are required. Therefore, part of the mentioned optimization logic might be moved to NNs running in a distributed manner on the devices of the edge infrastructure. The 5GCity Federated Machine Learning platform shall enable this, and thus in a performance-wise efficient manner.

4.1.1. Machine Learning with Resource-Constrained Devices

Mobile devices' resources are constrained in several aspects. Mobile devices such as smart phones are typically powered by batteries that have a relatively small amount of charge. In many cases, CPUs installed in those devices are battery efficient, but relatively slow. Mobile devices normally also come with wireless network interfaces. Those interfaces enable the devices to connect to 4G networks whose bandwidth is up to 100Mbps, with 5G networks expected to be 100 times faster than 4G ones. This shows one of the possible trade-offs between carrying out a computation locally on a slow CPU and consuming battery or shipping it to an edge node or data center, consuming power and bandwidth for transmission (not to mention the fact that the wireless throughout can be quite variable too).

Given this landscape, the research question is how and under which conditions to distribute NN workloads to edge and data center nodes. While existing research has already proposed a number of methods to distribute NN computation, these assume execution in a homogeneous, low delay and high bandwidth environment such as data-centers, where the trade-offs we have been mentioning largely do not exist. Given that federated ML presents a much more complex landscape, existing NN distribution techniques do not apply.

4.1.2. System Design

While there are several widespread NN frameworks (e.g., Theano, PyTorch, TensorFlow, etc), none of them support workload distribution, a requirement for an FML platform. As a result, we develop an FML platform that is able to split and distribute ML workloads to multiple different compute nodes. The FML platform is mainly composed of three components:

1. **The NN Directed Acyclic Graph (DAG) splitter**. NN workloads are usually expressed as DAGs. The FML platform cuts these DAGs into multiple shards so that they can be executed in different compute nodes in parallel.
2. **The RPC (Remote Procedure Call) protocol**. A shard is transferred to a compute node using an RPC protocol. We develop an RPC mechanism able to execute shards efficiently and flexibly.
3. **The task scheduler**. When multiple edge nodes are available, there is the executor selection problem. Proper worker scheduling is essential to achieving optimal performance. The FML platform implements a scheduling mechanism that properly balances workloads to multiple workers.

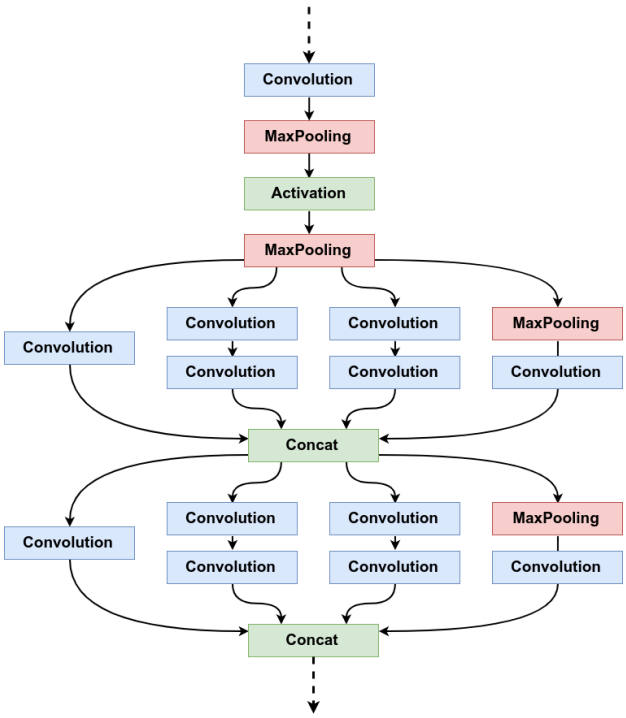

Figure 41. Example of CNN

As a target workload, we choose a Convolutional Neural Network (CNN), a popular network for image recognition. Figure 41 illustrates a part of a CNN. CNNs are constructed from convolution and pooling layers, among others. The DAG splitter cuts the DAGs of CNNs in several ways. The first way is model split. A CNN's layers are mainly matrix-based calculations. In model split, the DAG splitter splits a layer's matrix into multiple matrices. The second method is branch split. Some CNN models such as GoogleNet have branches in their DAGs. The DAG splitter cuts the DAG at the branch points. Figure 42 illustrates how model split and branch split work.

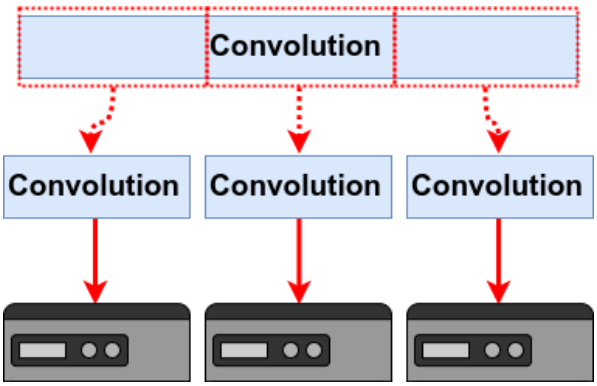

Figure 42. Model split.

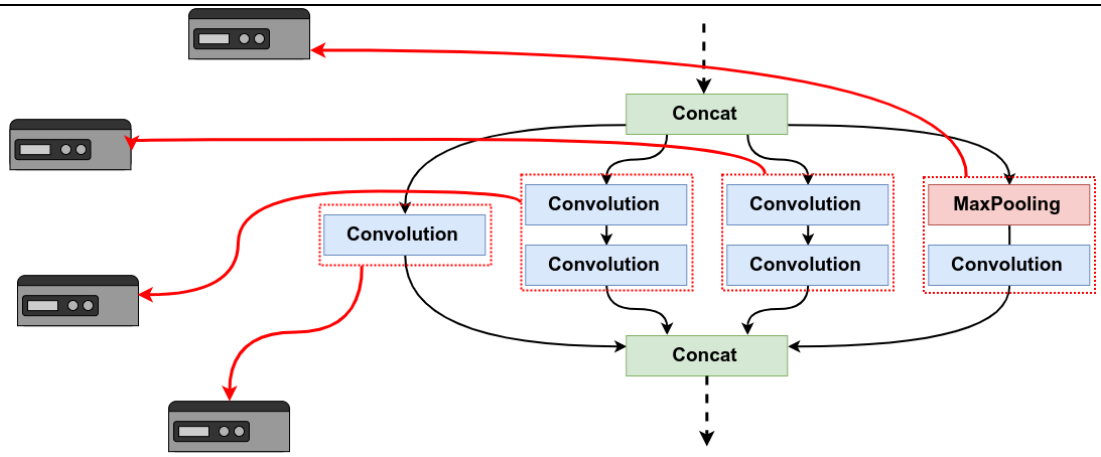

Figure 43. Branch split.

In the RPC mechanism of FML, there are 3 types of actors:

- **Worker node** are: in charge of the actual computation, a node runs an RPC server. When the RPC server receives a request, the worker node executes the requested workload and replies with the result.
- **Master nodes** are responsible for maintaining the information of worker nodes such as IP addresses and port numbers.
- **Clients** initiate the NN computation. First, a client node asks a master node for a list of available worker nodes. The master node provides a list of worker nodes to the client node. The client node the tries to establish RPC connections to worker nodes based on the received worker list. After the client node obtains the connections to worker nodes, the client starts to distribute its NN workload. The client node splits the NN workload by using the DAG splitter and submits them as requests to worker nodes.

While the FML platform enables mobile devices to distribute CNN computation to multiple edge nodes, there is still a worker selection problem. Because of the heterogeneity of resources described in the three-tier FML scenario, wrong worker selection can cause severely delays through the straggler problem, i.e., a master has to wait until the results from one slow node arrive, slowing down the entire computation. To deal with this, we developed a robust worker selection mechanism which copes with the heterogeneity of worker resources and dynamicity of network bandwidth. The worker scheduler component is implemented as a feature of the client node. The scheduling decision phase comes after the DAG splitter cuts the ML DAG into multiple shards which are treated as tasks submitted to workers. The basic scheduling algorithm adopted is first-in-first-out. The scheduler assigns a task to any possible worker as quick as possible. If there is no available worker when a new task is queued in the client node, the new task has to wait until a new worker becomes available. When a worker node finishes an assigned task, it replies the result to the client node and tells the worker node to become idle. Immediately after the client receives this notification, a queued task is submitted to the worker node. This mechanism automatically achieves optimal load-balancing since the scheduler assigns more tasks to fast workers than slow workers.

4.1.3. Performance Evaluation

We conduct a performance test in order to demonstrate the effectiveness of the FML platform. In this test, we measure how quickly the FML platform can process images. We compare the FML platform with the vanilla implementation of PyTorch. In the FML setup, we launch 12 worker nodes. We feed 12 images to MFL and measure the time spent for processing all images.

We carry out the experiment using a machine equipped with a 12-core CPU, Intel Xeon E5-2695 v2 2.4 GHz. Although the fundamental idea of FML is workload distribution, a fair comparison is impossible when the total available resources are not equal. Therefore, we conduct this benchmark in the same physical server. The maximum available resources are even in both cases, so this benchmark shows the fundamental benefit of FML.

The graph of Figure 44 shows the result of the benchmark. It took 7.3 sec for the baseline to process 12 images. On the other hand, it took 1.7 sec for the FML platform. Here we see 76.7% latency reduction. This performance gain comes from better CPU resource utilization. The baseline implementation (PyTorch) can utilize only up to 300% of CPU even though 1200% is the maximum. On the other hand, the FML platform achieves higher CPU utilization by distributing the NN workload to 12 workers running on different CPU cores. This result demonstrates the increased parallelism extends the amount of usable computation resources.

Finally, we note here that this improvement comes without the heterogeneity of computation performance between the local device and edge node. We expect further improvement when the performance difference between the local device and edge node is bigger than in this experiment; conducting such heterogeneous experiments is this subject of further work. We also aim, as future work, to conduct such tests using 5GCity's nodes and its multi-tier, federated architecture.

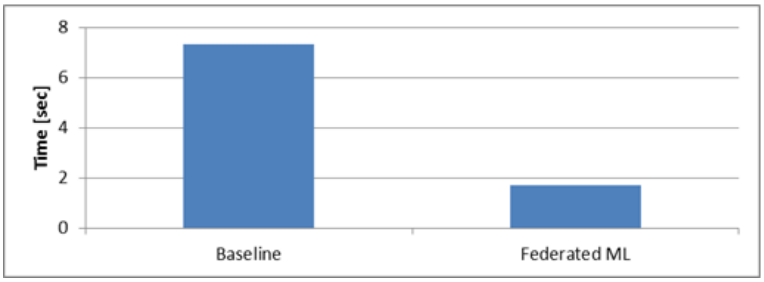

Figure 44. FML platform acceleration of image processing using PyTorch.

4.2. Using Artificial Intelligence in the 5GCity platform

The idea of using AI (Artificial Intelligence) in advanced communication networks is focused on automation reducing costs, increasing productivity, and driving more value. The main principle is to use AI for automating the operations processes based on collection and elaboration in real time of data about states and level of performances of systems and logical/virtualized resources etc. For instance, machine learning approach can automate the management, control and orchestration processes of physical pieces of equipment, which today are mostly carried out by humans, introducing control loops acting on virtual/logical entities (e.g., Virtual Machines, Containers, appliances etc.). Moreover, network and service computational intelligence (e.g., in the Radio Access Networks and in the Core), based on data about user service patterns and traffic would allow improving the quality of the user experience whilst optimizing the use of resources. Smart management mechanisms using machine learning at the service orchestration platform enable optimize the network operations experience towards automation and zero-touch orchestration. This will make the network more flexible allowing proactive resource allocation decisions based on heuristics rather than utilizing reactive approaches due to changes in the load. This approach is in line with the goal of a recent ETSI group called Experiential Network Intelligence (ENI) [19], which proposes an engine that adds closed-loop machine learning mechanisms based on context-aware and metadata-driven policies to more quickly recognize and incorporate new and changed knowledge, and hence, make automatically actionable decisions.

In this direction, the combination of network softwarization and AI will generate a huge amount of data, which is named, Big Data. The Big Data processing faces a number of challenges, which need to be addressed by the 5GCity architecture. The first and possibly most important challenge is scalability. While some systems have been able to adapt and handle easily terabytes of data in data centre environments, the data generation rate keeps growing, and whole set of new paradigms may have to be conceived. For instance, because its huge volume, it may not be feasible to move the data to a single location to be processed. Hence, the implementation of clouds may have to be extended to consider edge computing, and Big Data processing may have to be distributed. The distribution of the Big Data processing by using technologies like edge or fog computing has several important advantages over data centres. As an example, it allows the data to be processed near its source, reducing the amount of data that should traverse the communication networks.

4.3. Using Machine Learning for Scalable and Adaptive NFV

Due to the continuous development of SDN and NFV technology in recent years, it is important to improve the network performance to the end-users. In order to apply these two technologies (Adaptive and Autoscaling) in computer networks, we use SDN not only to separate the forwarding plane and control plane, but has the nature of the programmability also. Based on the actual business requirements for automatic deployment, NFV technology has the resources of virtualization and the characteristics of flexibility and fault isolation. Two kinds of technology are different, but they can work cooperatively very well. In 5GCity we are going to combine the SDN and NFV technology, to build an automated NFV ecosystem that include autonomous network management.

Network management currently undergoes massive changes towards realizing more flexible management of complex networks. Recent efforts include slicing data plane resources by using network (link) virtualization and applying operating system design principles in Software Defined Networking (SDN). Driven by network operators, network management principles are currently envisioned to be even further improved by virtualizing network functions, which are currently realized in dedicated hardware appliances. The resulting Network Function Virtualization (NFV) paradigm abstracts network functions from dedicated hardware to virtual machines running on commodity hardware and enables a Cloud-like network management. All of these efforts contribute to a softwarization of communication networks. This softwarization represents a significant change to network design and management by allowing the application of operating system

design and software engineering principles to make network management more efficient, e.g., by enabling flexible and dynamic service provisioning.

NFV offers the opportunity to run high-performance network services in a flexible way. Edge is a new place to provide such services, potentially with very low latency access for users. The types of "users" connecting to such services also may change over time not just users on phones or laptops, but autonomous vehicles, robots, smart city edge infrastructure, etc. These new types of users may require new types of NFV services, and may cause us to rethink the line between a network function and an application.

Critical issues to realize NFV in practice include proper performance evaluation methodologies towards predictable behaviour and in support of optimized VNF placement. In particular, network functions can be placed on-demand during an attack to inspect and filter traffic at multiple locations throughout the network. These locations can be dynamically chosen to be closer to the attacker sources as opposed to placing all filtering at the victim's network gateway where the attack traffic volume can already be intractable to control.

NFV together with SDN opens up new challenges for the composition, placement and migration of network function in an operator's network. This class of problems is generally referenced as the Function Placement Problem (FPP) inspired from the Controller Placement Problem that has been introduced for SDN controllers by Heller in HOTSDN 2012 [20]. However, there are challenges to deal with the FPP. First, as part of an optimal placement an optimal function (de-)composition and chaining has to be considered. SDN and NFV offer complementing concepts here where network functions can be moved completely (based on NFV) or partially (based on the SDN control/data plane split) into a data center [21]. Second, dynamic placement and migration must involve important network design aspects [22].

5. Conclusion

This Deliverable has specified the details of the parts of the 5GCity architecture that realize the orchestration, service programming, and edge-focused machine learning solutions. These are critical functionalities for satisfying the requirements of 5G Use Cases, especially of the neutral host scenario, and their implementation in the next phase of the 5GCity project should lead to important extensions of state-of-the-art Open Source solutions for orchestration and service programming.

The design of the related components, which is detailed in Sections 2, 3, and 4, has addressed –among others- the main state-of-the-art challenges that were identified in the Introduction, in the way summarized below.

CHALLENGE	HOW WE ADDRESS IT
1. The "slicing-unaware" and "Cloud-oriented" design of state-of-the-art NFV orchestrators restricts the efficiency of the management of neutral host scenarios by posing restrictions with regard to i) the number and the diversity of NFVI technologies that can be managed, ii) the degree of isolation of the orchestrated Network Services and VNFs of different stakeholders, iii) optimality of resource allocation and fragmentation, and iv) the diversity of captured and analysed runtime VNF parameters.	We eliminate the described restrictions by adjusting and extending the NFVO architecture (while also defining new Information Models) in a way that i) introduces an infrastructure abstraction layer between the NFVO and the NFVI ii) manages VNFs in the context of the custom and dynamic 5G slices upon which they will run, iii) incorporates efficient and 5G slicing-aware resource placement, and iv) connects the NFVO to MEC components, thus exploiting the existence of edge-related runtime VNF parameters that are part of the MEC information models..
2. For Service Programming, Service Development Kits (SDK) are a fundamental component to open-up the virtualization advantages. An SDK is a stand-alone collection of services, functionally integrated with an orchestration platform, able to craft network service templates ready to be deployed over a pool of virtual resources. Different SDK toolkits are already available within the NFV realm (e.g. the COHERENT SDK, the SONATA SDK, etc.). However, most of these SDKs adopt a network-centric approach, aiming at defining and testing Network Services and VNFs before their instantiation in runtime MANO environments. The user of those toolkits needs to be fully aware of the complex details of the information models used within the orchestration platform. Also those state-of-art toolkits are built on top of specific frameworks and bound to be used	An SDK toolkit has been designed to serve as a standalone entity, agnostic to the platform where the network services shall be deployed. This is done by means of the following main design ideas: (i) The 5GCITY SDK toolkit is composed of an adaptation layer which hides the complexity of 5GCity infrastructure low-level details. (ii) The 5GCITY SDK toolkit is designed to jointly work with a 5G Service and Application Catalogue, which serves as a point of exchange between SDK toolkit and orchestration platform, helping to translate from standard ETSI NFV information models to an information model which is specific to the orchestration platform where the network services need to be deployed.

in those specific deployments, thus missing the capability to encompass several NFV technologies.	
3. The state of the art in computing (whether learning or prediction) NNs is to assume that the infrastructure is entirely homogeneous, consisting of clusters of equally-spec'd servers with well-provisioned network links between them. Some of the previous challenges will be hard to handle without advanced intelligence, so the usage of AI might be essential. However, the right tools and learning strategies are not obvious.	Within the Federated Machine Learning solution we tackle the problem of running AI/NN computations in an environment that has wildly different hardware specs for the nodes, where energy consumption is a paramount factor (think mobile phones with batteries), and where network links are not only heterogeneous but can have huge differences in bandwidth over time, thus making AI an option for efficient resource placement in the 5GCity platform.

Appendix A. *Orchestration platforms state of the art analysis*

Tacker

Tacker is an official OpenStack project [3] that builds a Generic VNF Manager (VNFM) and a NFV Orchestrator (NFVO) to deploy and operate Network Services (NSs) and Virtual Network Functions (VNFs) on an NFV infrastructure platform like OpenStack (at this moment, the multi VIM is not supported, the tracker only support OpenStack). It is based on ETSI MANO Architectural Framework and provides a functional stack to Orchestrate Network Services end-to-end using VNFs, see Figure 45. It has capability to performs the basic life-cycle of VNF such as create, update and delete.

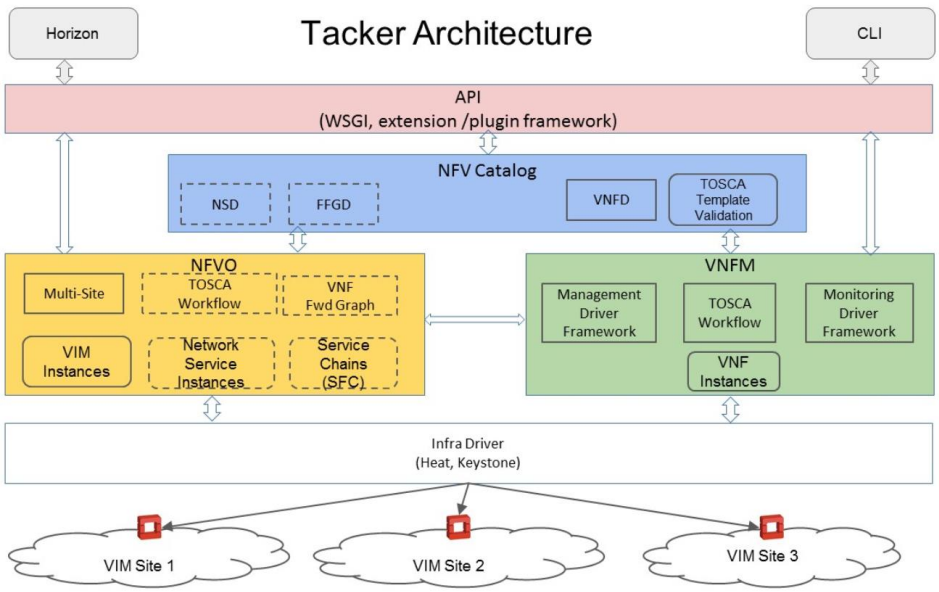

Figure 45. Tracker Architecture.

T-NOVA (TeNoR)

T-NOVA orchestrator has been named as TeNoR [4], which is ETSI-NFV compliant VNF deployment and operation. T-NOVA introduces a novel enabling framework, allowing operators to 1) deploy Virtualized Network Functions (VNFs) for their own needs, and 2) offer them to their customers, as value-added services. Virtual network appliances (gateways, proxies, firewalls, transcoders, analysers etc.) can be provided on-demand as-a-Service, eliminating the need to acquire, install and maintain specialized hardware at customers' premises.

For these purposes, T-NOVA designed and implemented a management/orchestration platform for the automated provision, configuration, monitoring and optimization of Network Functions-as-a-Service over virtualized Network/IT infrastructures. It leverages and enhances cloud management architectures for the elastic provision and (re-) allocation of IT resources assigned to the hosting of Network Functions. It also exploits and extends Software Defined Networking platforms for efficient management of the network infrastructure.

T-NOVA establishes a “NFV Marketplace”, in which network services and Functions by several developers can be published and brokered/traded. Via the Marketplace, customers can browse and select the services and virtual appliances which best match their needs, as well as negotiate the associated SLAs and be charged under various billing models. A novel business case for NFV is thus introduced and promoted. The T-NOVA architecture includes three main parts:

- Service Management and Life cycle concentrates all the features at the Network Service level.
- VNF Management and Life cycle concentrates all the features at the Virtual Network Function level.
- WIM (Wide-area network Interconnection Management) abstracts away all the interactions with the WAN (e.g., communication between VNFs that may leave in different DCs, connection to a specific customer’s network, etc.).

OpenBaton

Open Baton is an open source platform [5] that provides a comprehensive implementation of the ETSI NFV MANO specification. The main features and components of OpenBaton are 1) a Network Function Virtualisation Orchestrator (NFVO), 2) a generic Virtual Network Function Manager (VNFM) that manages VNF life cycles based on the VNF description, 3) An Auto-scaling Engine which can be used for automatic runtime management of the VNFs, 4) A Fault Management System for automatic management of faults, 5) an SDK comprising a set of libraries that could be used for building a specific VNFM, and 6) a dashboard for managing the VNFs.

The NFVO is the main component of OpenBaton, which is written in Java using the spring.io framework. To interconnect the NFVO to different VNFMs, OpenBaton relies on the Java Messaging System (JMS).

The NFVO is currently using OpenStack as first integrated NFV PoP VIM, supporting dynamic registration of NFV PoPs and deploys in parallel multiple slices one for each tenant, consisting of one or multiple VNFs. Through this functionality, the orchestrator provides a multi-tenant environment distributed on top of multiple cloud instances.

Cloudify

Cloudify is an open source TOSCA-based orchestration platform [6], which is designed to fit as an NFVO and a generic VNFM. Virtual Network Functions and Physical Network Function are modelled using the TOSCA language and on-boarded to Cloudify. Cloudify Model Driven Design allows operators to build VNF descriptors and Network Service Descriptors and manage the lifecycle of the network service. As illustrated in Figure 46, Cloudify Pluggable Architecture and the Plugin Framework makes integration to multiple VIMs and other platforms such as SDN Controllers, 3rd party VNFMs or hardware based system an easy and painless process. VNF modelling enables to describe the network service with all its resources: infrastructure, functions, service chaining, application code, scripts, configuration management, metrics, and policies, in a generic, descriptive language based on both TOSCA and Cloudify language.

The Orchestration is the core of the Cloudify Platform, it enables to maintain and run the complete life cycle of the service, from onboarding and instantiation to operations such as scaling, healing, maintenance, updates, and termination.

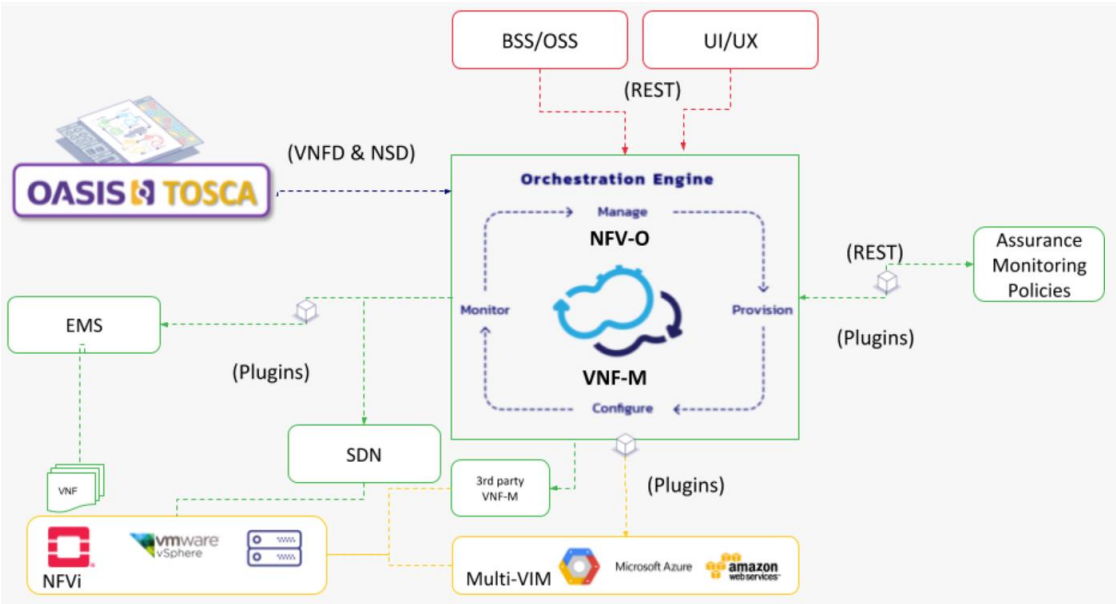

Figure 46. Cloudify MANO architecture.

5.1.1.1. Open Network Automation Platform (ONAP)

Open Network Automation Platform (ONAP) [7] provides a platform for real-time, policy-driven orchestration and automation of physical and virtual network function, which allow providers and developers to automate new services and support lifecycle management. Figure 47 shows a high-level architecture of the ONAP and its platform components.

The design time framework is a development environment with tools, techniques, and repositories for defining/describing resources, services, and products.

The runtime framework executes the rules and policies distributed by the design and creation environment. This framework distributes policy enforcement and templates among various ONAP modules such as the Service Orchestrator, Controllers, Data Collection, Analytics and Events, Active and Available Inventory, and a Security Framework.

The orchestration stack on ONAP provides for service delivery, change, scaling controller instantiation and capacity management across both the application and network controllers.

- The Service Orchestrator component is responsible for executing the specified processes and automates sequences of activities, tasks, rules and policies needed for on-demand creation, modification or removal of network, application or infrastructure services and resources.
- Controllers are applications that are coupled with cloud and network services and execute the configuration, real-time policies, and control the state of distributed components and services.
    - SDN-C: Cloud computing resource controller.
    - APP-C: Application controller.
    - VF-C: Virtual function controller (provide generic VNFM capability).
- Active and Available Inventory (A&AI) provides real-time views of a system's resources, services, products and their relationships with each other

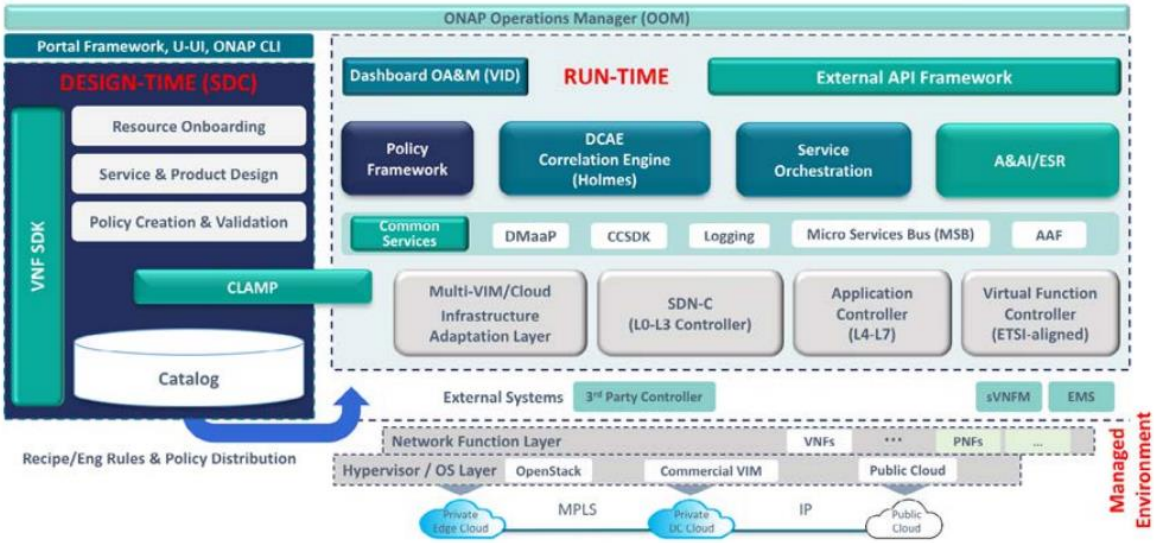

Figure 47. ONAP architecture.

ONAP is not support slicing yet, but it has plan to support the 5G specific services in the future.

SONATA

SONATA [8] is an orchestration platform based on ETSI MANO that provides development toolchain for virtualized services. The architectural level of SONATA includes two main components: service platform and SDK. The service platform offers customization on service platform operator and service developer. The SDK supports service developers with a programming model and a set of software tools. SONATA provides a modular and flexible MANO framework where a service or function specific manager can be added such as life-cycle management scaling, placement, and etc., thus modifying the provided default managers to a specific service or function needs.

As shown in Figure 48, the Service-Specific Managers (SSM) and Function-Specific Managers (FSM) are introduced in SONATA as main components by specifying desired placement or scaling behaviour for the service and function.

- SSM allows third-party service developers with control over specific orchestration and management functionalities pertaining to their own service.
- FSM Provides NFV MANO flexibility to network operators with customizable platform functionality and ability to add new features via plug-ins.

The Orchestrator provides a default manager for every network service (at the NFVO level) and VNF (at the VNFM level), but allows this generic behaviour to be adapted for each network service/VNF by their developers.

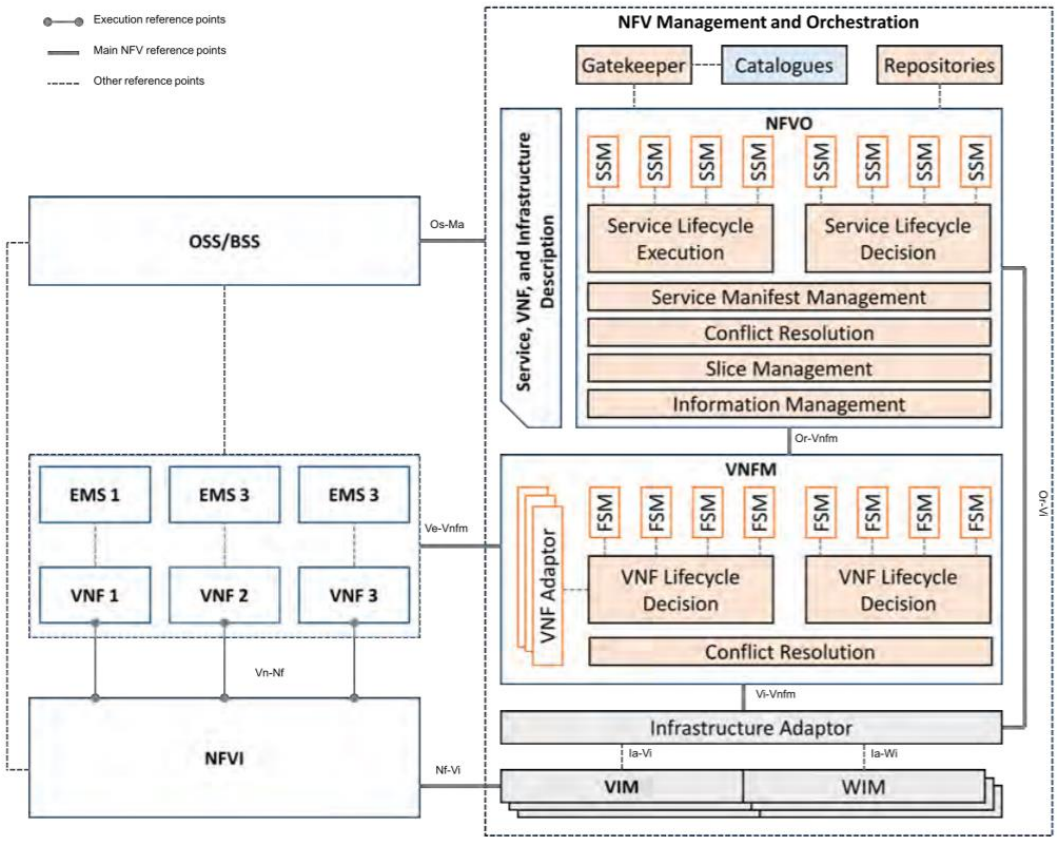

Figure 48. SONATA architecture mapped to ETSI MANO architecture.

SONATA architecture are:

- Support multi VIM, multi-vendor, and multi-site.
- Support both resource and service orchestration.
- Providing service developers with a SDK for efficient creation, deployment and management of VNF-based network services on the platform.
- NFV DevOps workflow and platform/SDK support bridges collaboration between operators and service developers.

Open Source MANO (OSM)

ETSI Open Source MANO (OSM) [9] is an operator-led ETSI community that is delivering a production-quality open source Management and Orchestration (MANO) stack aligned with ETSI NFV information models and that meets the requirements of production NFV networks.

The OSM community has set itself the goal of being an excellent production ready solution. OSM Release THREE represents another significant step along this path. It has been engineered, tested and documented to be functionally complete to support Operator RFx processes, and to be a key component for internal/lab and external/field trials as well as interoperability and scalability tests for virtual network functions and services.

The OSM group has defined an expansive scope for the project covering both design-time and run-time aspects related to service delivery for telecommunications service provider environments. Figure 49 shows the approximate mapping of the OSM components to the ETSI NFV MANO architecture framework.

The Design-Time Framework architecture are:

- A model-driven environment with data mode aligned with ETSI NFV MANO.

- The capability of create, read, update and delete operations on the Network Service Definition.
- A VNF package generation.
- A GUI to accelerate network service design time phase, VNF on-boarding and deployment.

The Run-Time Framework architecture includes:

- An automated Service Orchestration environment that enables and simplifies the operational considerations of the various lifecycle phases involved in running a complex service based on NFV.
- A superset of ETSI NFV MANO where the salient additional area of scope includes Service Orchestration but also explicitly includes provision for SDN control.
- A plugin model for integrating multiple SDN controllers.
- A plugin model for integrating multiple VIMs, including public cloud based VIMs.
- A plugin model for integrating multiple monitoring tools into the environment.
- A reference VIM that has been optimized for Enhanced Platform Awareness (EPA) to enable high performance VNF deployments.
- An integrated "Generic" VNFM with support for integrating "Specific" VNFMs.
- An integrated Physical Network Functions into an automated Network Service deployment.
- Support both Greenfield and Brownfield deployment scenarios.
- GUI and CLI, a Python based client library and REST interfaces to enable access to all features.

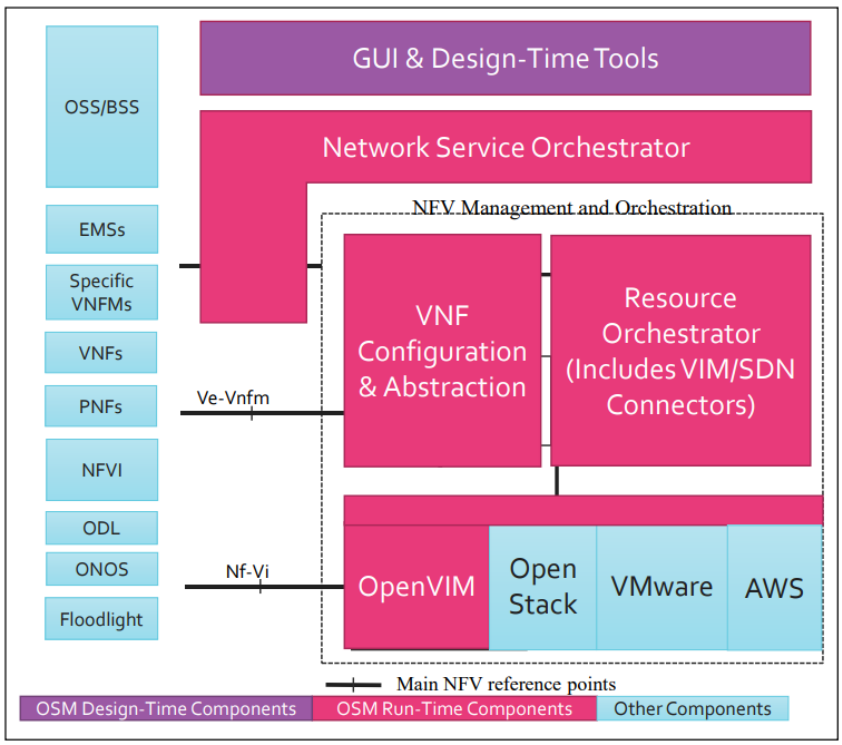

Figure 49. OSM architecture mapping to ETSI framework.

The OSM orchestration solution is covering all the aspects of network service life cycle management and network function life cycle management. Figure 50 shows the logical blocks functionality delivered by the OSM.

- The Service Orchestration Engine is responsible for all aspects of service orchestration including lifecycle management and service primitive execution. It is effectively the "master" orchestration component in the system that governs the workflow throughout OSM. In addition, it supports the concepts of multi-tenancy, projects, users, and enforcing role-based access controls.

- The Configuration Data Store is responsible for persistently storing the SO state, particularly in the context of VNF and NSD deployment records.
- The Network Service Composition Engine is responsible for supporting NS and VNF descriptor composition. It validates the composed NS and VNF descriptors conform to the defined YANG schema.
- The Catalog Manager is responsible for supporting the “create, read, update, delete” lifecycle operations on the defined VNF and NS descriptors and packages.
- The Resource Orchestrator Plugin provides an interface to integrate the Resource Orchestrator (RO).
- The Network Service to VNF Communication (N2VC) plugins framework between the Service Orchestrator (SO) and the VNF Configuration and Abstraction (VCA) layer.
- The VNF Configuration and Abstraction (VCA) layer enables configurations, actions and notifications to/from the VNFs and/or Element Managers.
- The Resource Orchestration Engine manages and coordinates resource allocations across multiple geo-distributed VIMs and multiple SDN controllers.

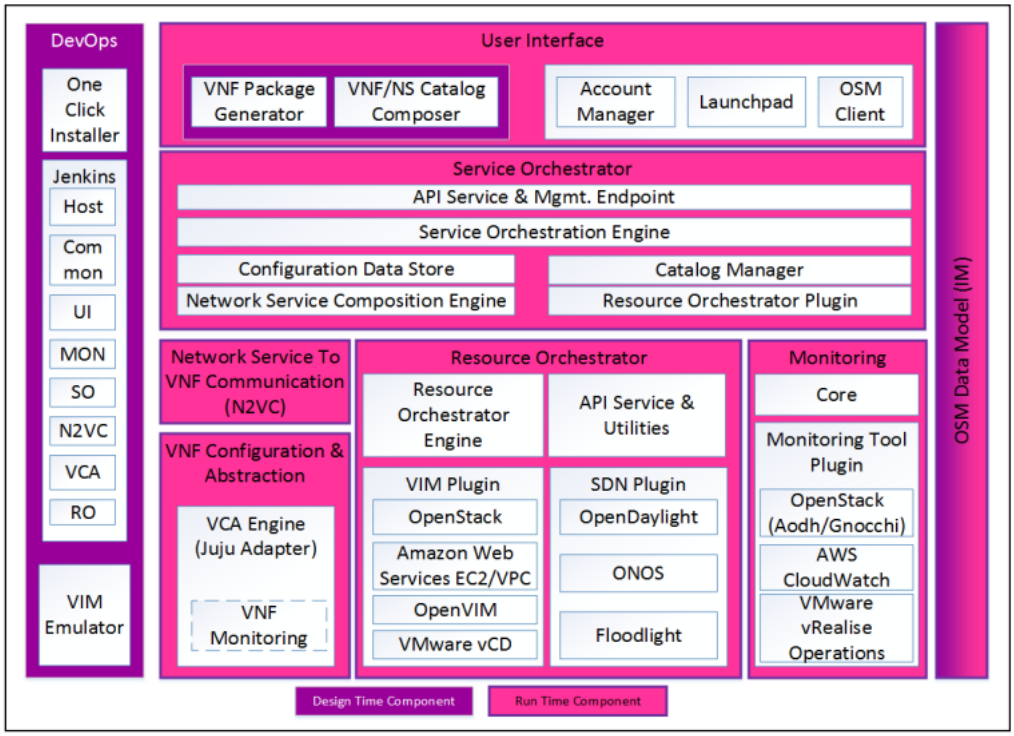

Figure 50. OSM release three architecture.

As slicing in not still covered in the latest version of the OSM (release three), the OSM community created an official OSM Proof of Concept (PoCs) framework to identify opportunities for further development of OSM. The PoC has been used to introduce the application of DevOps for supporting network slicing in the OSM.

Appendix B. *Related Work for Resource Placement*

Cloud computing offers computing resources as a utility: one no longer has to manage and maintain its own private computing infrastructure, instead computing infrastructure is time-shared and can run applications on-demand.

Nowadays, there is a large growth of a specific class of applications that has stringent requirements, such as low latency, high processing demand, and large bandwidth. This class of applications are known as applications with 5G requirements.

Although most applications do not have specific requirements, this class of applications demands that Cloud technology allows elastic scaling of resources when demand for an application changes.

To meet these specific requirements, Data Centers (DC) grow in size and complexity, hosted applications become increasingly vulnerable to dynamically occurring cloud infrastructure bottlenecks. As an example of this complexity, cloud infrastructures are built hierarchically geographically distributed. In this way, services that require low latency should be executed in the edge, while services that need high computational power should be allocated in centralized DCs.

Additionally, while heterogeneity in a DC is limited to multiple generations of servers being used, there is a large spread on capabilities within a cloud environment. Memory and processing means range from high (e.g. servers), over medium (e.g. cloudlets, gateways) to very low (e.g. mobile devices, sensor nodes). While some communication links guarantee a certain bandwidth (e.g. dedicated wired links), others provide a bandwidth with a certain probability (e.g. a shared wired link), and others do not provide any guarantees at all (wireless links).

Reliability is an important non-functional requirement, as it outlines how the software systems realizes its functionality [23]. The unreliability of substrate resources in a heterogeneous cloud environment, severely affects the reliability of the applications relying on those resources. Therefore, it is very challenging to host reliable applications on top of unreliable infrastructure [24].

Applications are composed by services, and to allocate these services to meet the requirements of each of the applications, on-demand clouds use placement algorithms. There are potentially two types of service placement decisions to be made: initial placement [25] and migration (and/or resizing) of service resources over time [26], as the available resources are changed or new demands emerge from the applications new placement decisions must be taken.

The problem of mapping the services in physical resources, whilst considering failures in the cloud environment are known as Survivable Virtual Network Embedding (SVNE) [27]. The goal of allocation algorithms is to satisfy a minimum level of total availability for each application in case of cloud infrastructure failure.

There are different service placement algorithms [28]. Early work merely considers nodal resources, such as Central Processing Unit (CPU) and memory capabilities. Deciding whether requests are accepted and where those virtual resources are placed then reduces to a Multiple Knapsack Problem (MKP) [29]. An MKP is known to be NP-hard and therefore optimal algorithms are hampered by scalability issues. A large body of work has been devoted to finding heuristic solutions. For instance, [30] focuses on the multi-objective Virtual Machines (VMs) placement problem. They propose a genetic algorithm with fuzzy multi-objective

evaluation for efficiently searching the large solution space and conveniently combining possibly conflicting objectives. While Yi et al. propose an evolutionary game theoretic framework for adaptive and stable application deployment in clouds [31]. Other works include Network Interface Card (NIC) capabilities as a dimension in the MKP [32] and assumes an over-provisioned inner-network. While plausible within the boundaries of one DC, this condition rarely holds when a combination of multiple clouds or even a wireless environment is considered.

In this field the application requests are usually modelled as Virtual Networks (VNs), and consisting of services and their required communication channels. The cloud infrastructure is modelled as a Substrate Network (SN) consisting of physical nodes and their interconnecting links. The problem of mapping the VNs to a SN, whilst considering failures in the SN is known as the Survivable Virtual Network Embedding (SVNE) problem.

When the application placement not only decides where computational entities are hosted, but also decides on how the communication between those entities is routed in the SN, then we speak of network aware APP. Network-aware application placement is closely tied to Virtual Network Embedding (VNE) [33].

An example of a network-aware approach is the work from [34]. It employs a Service Oriented Architecture (SOA), in which applications are constructed as a collection of communicating services. This optimal approach performs node and link mapping simultaneously. In contrast, other works try to reduce computational complexity by performing those tasks in distinct phases [35].

While the traditional VNE problem assumes that the SN network remains operational at all times, the SVNE problem does consider failures in the SN. For instance, Ajtai et al. try and guarantee that a virtual network can still be embedded in a physical network, after k network components fail. They provide a theoretical framework for fault-tolerant graphs [36]. However, in this model, hardware failure can still result in service outage as migrations may be required before normal operation can continue.

Csorba et al. propose a distributed algorithm to deploy replicas of VM images onto PMs that reside in different parts of the network [37]. The objective is to construct balanced and dependable deployment configurations that are resilient. The problem is that the number of replicas to be placed is assumed predefined.

Chowdhury et al. propose Dedicated Protection for Virtual Network Embedding (DRONE) [38]. DRONE guarantees VN survivability against single link or node failure, by creating two VNEs for each request. These two VNEs cannot share any nodes and links.

The aforementioned SVNE approaches lack an availability model. When the infrastructure is homogeneous, it might suffice to say that each VN or VNE need a predefined number of replicas. However, in geo-distributed cloud environments the resulting availability will largely be determined by the exact placement configuration, as moving one service from an unreliable node to a more reliable one can make all the difference. Therefore, geo-distributed cloud environments require SVNE approaches which have a computational model for availability as a function of SN failure distributions and placement configuration.

The following cloud management algorithms have a model to calculate availability: Jayasinghe et al. [39] model cloud infrastructure as a tree structure with arbitrary depth. Physical hosts on which VMs are hosted are the leaves of this tree, while the ancestors comprise regions and availability zones. The nodes at bottom level are physical hosts where VMs are hosted. Wang et al. [40] were the first to provide a mathematical model to estimate the resulting availability from such a tree structure. They calculate the availability of a single VM as the probability that neither the leaf itself, nor any of its ancestors fail. Their work focuses on handling workload variations by a combination of vertical and horizontal scaling of VMs. Horizontal scaling launches or suspends additional VMs, while vertical scaling alters VM dimensions. The total availability is then the probability that at least one of the VMs is available. While their model suffices for traditional

clouds, it is illsuited for a geo-distributed cloud environment as link failure and bandwidth limitations are disregarded.

In contrast, Yeow et al. define reliability as the probability that critical nodes of a virtual infrastructure remain in operation over all possible failures [41]. They propose an approach in which backup resources are pooled and shared across multiple virtual infrastructures. Their algorithm first determines the required redundancy level and subsequently performs the actual placement. However, decoupling those two operations is only permissible when link failure can be omitted and nodes are homogeneous.

In [24], an availability model for geo-distributed cloud networks was introduced, which considers any combination of node and link failures, and supports both node and link replication. The aforementioned model was employed to study the problem of guaranteeing a certain level of availability for applications. Using an ILP formulation of the problem and an exact solver, an increased placement ratio was demonstrated, compared to naive approaches which lack an availability model. While the ILP solver can find optimal placement configurations for small scale networks, its computation time quickly becomes unmanageable when the substrate network dimensions increase. In [42] a first heuristic is presented. This distributed evolutionary algorithm employs a pool model, where execution of computational tasks and storage of the population database (DB) are separated.

Many of these algorithms can be used by available open source orchestration tools such as Tacker and OSM, however such implementations may not be flexible enough that would require modifications to the packages as can be seen in [43] and [44].

Abbreviations and Definitions

3GPP. *3d Generation Partnership Project*
AI. *Artificial Intelligence*
API. Application Programming Interface
CNN. *Convolutional Neural Network*
CPU. *Computer Processing Unit*
DAG. *Directed Acyclic Graph*
DC. *Data Center*
ENI. *Experiential Network Intelligence*
ETSI. *European Telecommunications Standards Institute*
EU. *European Union*
FBP. *Flow-Based Programming*
FML. *Federated Machine Learning*
FPP. *Function Placement Problem*
FSM. *Function-Specific Managers*
GPU. *Graphics Processing Unit*
IoT. *Internet of Things*
KPI. *Key Performance Indicator*
LCM. *LifeCycle Management*
LTE. *Long Term Evolution*
MANO. *Management and Orchestration*
ME. *Multi-access Edge*
MEAO. *Multi-access Edge Application Orchestrator*
MEC. *Multi-access Edge Computing*
MEPM-V. *Multi-access Edge Platform Manager – Virtual*
ML. *Machine Learning*
N2VC. *Network Service to VNF Communication*
NFV. *Network Function Virtualization*
NFVI. *Network Function Virtualization Infrastructure*
NFVO. *Network Function Virtualization Orchestrator*
NN. *Neural Network*
NS. *Network Service*
NSD. Network Service Descriptor
ODL. *OpenDayLight*
ONAP. *Open Network Automation Platform*
OSS/BSS. *Operations Support System / Billing Support System*
PNF. *Physical Network Function*
PNFD. *Physical Network Function Descriptor*
PoC. *Proof of Concept*
PoP. *Point of Presence*
QoS. *Quality of Service*
RBAC. *Role Based Access Control*
REST. *REpresentational State Transfer*
RO. *Resource Orchestrator*
RPC. *Remote Procedure Call*

RRH. *Remote Radio Head*
RU. *Radio Unit*
SDK. *Service Development Kit*
SDN. *Software-Defined Networking*
SLA. *Service Level Agreement*
SO. *Service Orchestrator*
SSM. *Service-Specific Managers*
VCA. *VNF Configuration and Abstraction*
vCPU. *virtual Computer Processing Unit*
VDU. *Virtualisation Deployment Unit*
vHDD. *virtual Hard Disk Drive*
VIM. *Virtualized Infrastructure Manager*
VL. *Virtual Link*
VLD. *Virtual Link Descriptor*
VM. *Virtual Machine*
VNF. *Virtual Network Function*
VNFD. *Virtual Network Function Descriptor*
VNFFG. *Virtual Network Function Forwarding Graph*
VNO. *Virtual Network Operator*
vRAM. *virtual Random Access Memory*
WIM. *Wide-area network Interconnection Management*

References

[1] 5GCity Project, Deliverable 2.2: 5GCity Architecture and Interfaces Definition. Online at https://www.5gcity.eu/deliverables/, 2018.

[2] 5GCity Project, Deliverable 2.1: 5GCity System Requirements and Use Cases. Online at https://www.5gcity.eu/deliverables/, 2018.

[3] Tracker, OpenStack NFV Orchestration. Online at https://wiki.openstack.org/wiki/Tacker.

[4] TeNoR, TeNoR – T-nova Orchestration platform. Online at http://www.t-nova.eu/open-source/.

[5] Open Baton, An Open source reference implementation of the ETSI NFV MANO. Online at https://openbaton.github.io/.

[6] Cloudify, Cloud and NFV Orchestration based on TOSCA. Online at https://cloudify.co/.

[7] ONAP, Open Network Automation Platform. Online at https://www.onap.org/.

[8] SONATA Project, Online at http://www.sonata-nfv.eu/.

[9] ETSI, Open Source MANO (OSM). Online at https://osm.etsi.org/.

[10] ETSI, Mobile Edge Computing (MEC) - Deployment of Mobile Edge Computing in an NFV environment. ETSI GR MEC 017 V1.1.1, 2018.

[11] 5GCity Project, Deliverable 3.1: 5GCity Edge Virtualization Infrastructure Design. Online at https://www.5gcity.eu/deliverables/, 2018.

[12] Fog05, Fog Computing IaaS. Online at http://www.fog05.io.

[13] OpenStack, Online at https://www.openstack.org/.

[14] ETSI, “Network Functions Virtualisation (NFV) Release 2; Management and Orchestration; Network Service Termplates Specification. ETSI GS NFV-IFA 014 V2.4.1,” 2018.

[15] ETSI, “Network Functions Virtualisation (NFV) Release 2; Management and Orchestration; VNF Descriptor and Packaging Specification, ETSI GS NFV-IFA 011 V2.4.1,” 2018.

[16] Prometheus, Monitoring System and Time Series Database. Online at https://prometheus.io/.

[17] J. P. Morrisson, Flow-Based Programming, 2nd Edition: A New Approach to Application Development. CreateSpace, Paramount, 2010.

[18] ETSI, Network Functions Virtualisation (NFV) Release 2; Management and Orchestration; Os-Ma-Nfvo reference point - Interface and Information Model Specification ETSI GS NFV-IFA 013 V2.4.1, 2018.

[19] ETSI, Experiential Network Intelligence Whitepaper. Online at http://www.etsi.org/images/files/ETSIWhitePapers/etsi_wp22_ENI_FINAL.pdf., 2017.

[20] B. Heller, R. Sherwood and N. McKeown, "The controller placement problem," in *ACM first workshop on Hot topics in software defined networks (HotSDN '12)*, 2012.

[21] A. Basta, W. Kellerer, M. Hoffmann, H. Morper and K. Hoffmann, "Applying NFV and SDN to LTE Mobile Core Gateways; The Functions Placement Problem," in *ACM SICGOMM Workshop AllThingsCellular14*, 2014.

[22] A. Basta, A. Blenk, M. Hoffmann, H. Morper, K. Hoffmann and W. Kellerer, "SDN and NFV Dynamic Operation of LTE EPC Gateways for Time-varying Traffic Patterns," in *6th International Conference on Mobile Networks and Management (MONAMI)*, 2014.

[23] ISO, "ISO/IEC-25010, Systems and software engineerng. Systems and software Quality Requirements and Evaluation (SQuaRE). System and software quality models," 2010.

[24] B. Spinnewyn and S. Latré, "Towards a fluid Cloud: An extnsion of the Cloud into the local Network.," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), Vol. 9122, Gh*, 2015.

[25] M. Cardosa, A. Singh, H. Pucha and A. Chandra, "Exploiting Spatio-Temporal Tradeoffs for Energy Efficient MapReduce in the Cloud," in *Department of Computer Science and Engineering, University of Minnesota, TR-10-008*, 2010.

[26] M. Chen, H. Zhang, Y.Y. Su, X. Wang, G. Jiang and K. Yoshihra, "Effective VM Sizing in Virtualized Data Centers," in *12th IFIP/IEEE Symposium on Integrated Network Management*, 2011.

[27] M. Rahman and R. Boutaba, "SVNE: survivable virtual network embedding algorithms for network virtualization.," in *IEEE Transactions on Network and Service Management 10 (2), Pages 105-118*, 2013.

[28] B. Spinnewyn, R. Mennes, J. F. Botero and S. Latré, "Resilient Application Placement for Geo-distributed Cloud Networks," in *Journal of Network and Computer Applications, Volume 85, Pages 14-31*, 2017.

[29] R. Camati and A. Calsavara, "Solving the virtual machine placement problem as a multiple multidimensional knapsack problem," in *ICN 2014, Pages 253–260*, 2014.

[30] J. Xu and J. Fortes, "Multi-objective virtual machie placementin virtualized data center environments.," in *IEEE/ACM International Conference on Cyber, Physical and Social Computing (CPSCom), GREENCOM-CPSCOM*, 2010.

[31] Y. Ren, J. Suzuki, A. Vasilakos, S. Omura and K. Oba, "Cielo: An evolutionary game teoretic framework for virtual machine placement in clouds.," in *International Conference on Future Internet of Things and Cloud, FiCloud 2014, Pages 1-8*, 2014.

[32] H. Moens, E. Truyen, S. Walraven, W. Joosen, B. Dhoedt and F. De Turck, “CostEffective feature placemnt of customizable multi-tenant applications in the cloud.,” in *Journal of Network Systems Management 22 (4), Pages 517–558*, 2014.

[33] A. Fischer, J. Botero, M. Beck, H. De Meer and X. Hesselbach, “Virtual network embedding: a survey,” in *IEEE Communications Surveys and Tutorials 15 (4), Pages 1888–1906*, 2013.

[34] H. H. B. D. B. D. T. F. Moens, “Hierarchical network-aware plcement of service oriented applications in clouds,” in *IEEE/IFIP Network Operations and Management Symposium: Management in a Software Defined World, Pages 1-8*, 2014.

[35] X. Cheng, S. Su, Z. Zhang, H. Wang, F. Yang, Y. Luo and J. Wang, “Virtual network embedding through topology-aware node ranking,” in *ACM SIGCOMM Computing and Communications Review 41 (2)*, 2011.

[36] M. Ajtai, N. Alon, J. Bruck, R. Cypher, C. Ho, M. Naor and E. Szemeredi, “Fault tolerant graphs, perfect hash functions and disjoint paths,” in *33rd Annual Symposium on Foundations of Computer Science, Pages 693–702*, 1992.

[37] M. Csorba, H. Meling and P. Heegaard, “Ant system for service deployment in private and public clouds,” in *2nd workshop on Bio-inspired algorithms for distributed systems (BADS '10)*, 2010.

[38] S. Chowdhury, R. Ahmed, M. Alam Khan, N. Shahriar, R. Boutaba, J. Mitra and F. Zeng, “Dedicated Protection for Survivable Virtual Network Embedding,” in *IEEE Transactions on Network and Service Management*, 2016.

[39] D. Jayasinghe, C. Pu, T. Eilam, M. Steinder, I. Whalley and E. Snible, “Improving performance and avalability of services hosted on IaaS clouds with structural constraint-aware virtual machine placement,” in *IEEE International Conference on Services Computing (SCC '11), Pages 72-79*, 2011.

[40] W. Wang, H. Chen and X. Chen, “An availability-aware virtual machine placement approach for dynamic scaling of cloud applications,” in *IEEE 9th International Conference on Ubiquitous Intelligence and Computing and IEEE 9th International Conference on Autonomic and Trusted Computing (UIC-ATC 2012)*, 2012.

[41] W.-L. Yeow, C. Westphal and U. Kozat, “Designing and embedding reliable virtual infrastructures,” in *2nd ACM SIGCOMM workshop on Virtualized infrastructure systems and architectures (VISA '10), 41 (2)*, 2010.

[42] R. Mennes, B. Spinnewyn, S. Latré and J. Botero, “GRECO: A Distributed Genetic Algorithm for Reliable Application Placement in Hybrid Clouds,” in *IEEE Proceedings of the 5th International Conference on Cloud Networking (CloudNet '16)*, 2016.

[43] J. Chen, Y. Chen, S.-C. Tsai and Y.-B. Lin, “Implementing NFV System with OpenStack,” in *IEEE Conference on Dependable and Secure Computing*, 2017.

[44] M. Kourtis, M. McGrath, G. Gardikis, G. Xilouris, V. Riccobene, P. Papadimitriou, E. Trouva, F. Liberati, M. Trubian, J. Batalle, H. Koumaras, D. Dietrich, A. Ramos, J. F. Riera, J. Bonnet, A. Pietrabissa, A. Ceselli and A. Petrini, “T-NOVA: An Open-Source MANO Stack for NFV Infrastructures,” in *IEEE Transactions on Networks and Services Management, vol. 14, no. 3, Pages 586–602*, 2017.

<END OF DOCUMENT>