Deliverable D4.1

Plug & Play Control Plane for Sliced Networks

Editor:	Giacomo Bernini, Nextworks
Deliverable nature:	Report (R)
Dissemination level:	PUBLIC (PU)
Contractual delivery date:	30/09/2018
Actual delivery date:	16/10/2018
Suggested readers:	Network Administrators, DevOps (Development and Operations) Professionals, Telecommunication Operators, Service Providers
Version:	1.0
Total number of pages:	111
Keywords:	Plug & Play, vertical-in-the-loop, slice exposure, microservices, containers

Abstract

This document reports the design and prototype implementation of the SliceNet Plug & Play control framework. It represents the enabling technology for providing verticals and slice consumers with tailored control exposure and view of their slices. In particular, the SliceNet Plug & Play allows to deploy a dedicated control environment for each slice exposed to verticals, that is customized in order to offer tailored control capabilities that meet the vertical requirements in terms of runtime control. This is enabled by a highly dynamic, flexible and extensible Plug & Play design approach based on microservices. Each Plug & Play control instances is a collection of microservices providing the required slice abstraction and tailored view leveraging on a generalized slice control exposure information model, while customizing the runtime control operations to be exposed to the verticals. Specific control functions and plugins microservices enable the exposure of pure slice control, as well as monitoring and management capabilities to verticals. From a software prototype perspective, the Plug & Play microservices are implemented as containerized applications that are orchestrated with an open source tool like Kubernetes.

Disclaimer

This document contains material, which is the copyright of certain SliceNet consortium parties, and may not be reproduced or copied without permission.

All SliceNet consortium parties have agreed to full publication of this document.

The commercial use of any information contained in this document may require a license from the proprietor of that information.

Neither the SliceNet consortium as a whole, nor a certain part of the SLICENET consortium, warrant that the information contained in this document is capable of use, nor that use of the information is free from risk, accepting no liability for loss or damage suffered by any person using this information.

The EC flag in this document is owned by the European Commission and the 5G PPP logo is owned by the 5G PPP initiative. The use of the flag and the 5G PPP logo reflects that SliceNet receives funding from the European Commission, integrated in its 5G PPP initiative. Apart from this, the European Commission or the 5G PPP initiative have no responsibility for the content.

The research leading to these results has received funding from the European Union Horizon 2020 Programme under grant agreement number H2020-ICT-2014-2/761913.

Impressum

[Full project title] End-to-End Cognitive Network Slicing and Slice Management Framework in Virtualised Multi-Domain, Multi-Tenant 5G Networks

[Short project title] SliceNet

[Number and title of work-package] WP4 - 5G Multi-Domain Slice Control Plane

[Number and title of task] T4.1 - Plug & Play Control Plane for Sliced Networks

[Document title] Plug & Play Control Plane for Sliced Networks

[Editor] Giacomo Bernini, Nextworks

[Work-package leader:] Ciriaco Angelo, Ericsson

Copyright notice

Executive summary

SliceNet aims at defining a vertical-oriented, Quality of Experience (QoE)-driven network slicing framework that focuses on cognitive network management and control of end-to-end network slicing operation and slice-based (and slice-enabled) services in 5G networks. With this in mind, the delivery of heterogeneous services customized for a large set of vertical businesses with diverging requirements (e.g. in terms of Quality of Service (QoS), QoE and Service Level Agreement) requires an effective orchestration, management and control of 5G networks able to keep verticals in the loop of services and slices provisioning, from design to operation.

However, current offerings from service providers and network operators are still based on a very limited involvement of verticals and consumers in the runtime control and management of the offered services. SliceNet intends to fill this gap along the development of a network slicing approach, by allowing vertical actors to have management access to their slices and apply tailored control logics on top of them through a set of customized control operations. In practice, the Plug & Play control framework is the key enabler for this evolution of services and slice control exposure towards a paradigm where verticals play an active role in the runtime control and management of their slices.

This deliverable defines the SliceNet Plug & Play control framework that allows verticals to be provided with isolated control environments where customized control functions and logics are plugged to build a truly tailored slice view. In this context, the Plug & Play control is able to accommodate heterogeneous vertical expectations and requirements for slice exposure, in terms of topological view and runtime operations for controlling, managing and monitoring slice instances. In practical terms, it provides a flexible, common runtime control framework that can be easily customized to expose vertical-tailored slice control views and operations. To enable this, the Plug & Play control follows a microservices design approach. Each Plug & Play control instance is a collection of self-contained microservices with each one contributing a specific logic to an overall customized slice view, including and control functions microservices implementing specialized control logics. This approach enables a vertical-tailored slice runtime customization paradigm to evolve along two main directions: i) how the slice is exposed (in terms of slice components and topology view), ii) how the slice and its components can be controlled and managed at runtime by the vertical. This Plug & Play control approach lays its customization principle on a technology-agnostic slice exposure information model, where a given slice instance is represented in the form of a generalized topology graph that allows customization of each element in the graph through the association with a slice-context. In addition, each element in the graph is also augmented with the set of control operations and endpoints (including management and monitoring when applicable) as those are foreseen to be exposed to the vertical by the Plug & Play control synthesis. On top of this a set of northbound APIs are intended to be mediated (to provide at least authentication and role based access features) by the One-Stop-API layer. These APIs are not based on a pre-determined set of endpoints and operations, but they are indeed dynamically prepared and exposed by each Plug & Play control instance according to the tailored slice view to be exposed (as instantiation of the generalized information model) and the combination of plugins and control functions microservices deployed.

The Plug & Play control framework software prototype has been developed from scratch, and it is described in terms of APIs, information models, release format, installation and operation guidelines. From a software and deployment perspective, each Plug & Play control instance is a collection of Docker containers deployed, coordinated and orchestrated by Kubernetes. This allows an extremely agile, flexible and scalable Plug & Play deployment approach that leverages on de-facto standard open source tools for running and orchestrating Plug & Play control instances as containerized applications.

List of authors

Institution	Author
NXW	Giacomo Bernini, Pietro G. Giardina, Gino Carrozzo, Elian Kraja
CSE	Konstantinos Koutsopoulos
UPC	Fernando Agraz, Albert Pagès, Rafael Montero, Salvatore Spadaro
ALB	Pedro Neves, José Cabaça
TEI	Ciriaco Angelo, Luca Baldini, Anna Maria Santonicola

Deliverable Reviewers

Reviewer	Institution
Qi Wang	UWS
Marius Iordache	ORO

Table of Contents

List of tables

List of figures

Abbreviations

API	Application Programming Interface
BBU	Base Band Unit
CN	Core Network
CP	Control Plane
CQI	Channel Quality Indicator
CQI	Channel Quality Indicator
CSP	Communication Service Providers
DSC	Digital Service Customer
DSP	Digital Service Provider
E2E	End-to-End
EPC	Evolved Packet Core
ETSI	European Telecommunications Standards Institute
ID	Identifier
IETF	Internet Engineering Task Force
IMSI	International Mobile Subscriber Identity
KPI	Key Performance Indicator
LCM	Lifecycle Management
MANO	Management and Orchestration
ME	Mobile Edge
MEC	Mobile/Multi-access Edge Computing
MEO	Mobile Edge Orchestrator
NE	Network Element
NEF	Network Exposure Function
NF	Network Function
NFV	Network Function Virtualisation
NFVI	NFV Infrastructure
NS	Network Service; Network Slice
NSaaS	Network Slice as a Service
NSD	Network Service Descriptor
NSEP	Network Slice Provider
NSI	Network Slice Instance
NSP	Network Service Provider
NST	Network Slice Template

OAM	Operations, administration and maintenance
OSS	Operations Support System
P&P	Plug & Play
PNF	Physical network Function
PoP	Point-of-Presence
QoE	Quality of Experience
QoS	Quality of Service
REST	Representational State Transfer
RO	Resource Orchestrator
RRH	Remote Radio Head
SDN	Software Defined Networks
SDO	Standards Developing Organization
SLA	Service Level Agreement
SliceNet	End-to-End Cognitive Network Slicing and Slice Management Framework in Virtualised Multi-Domain, Multi-Tenant 5G Networks
SS-O	Slice Service Orchestrator
UC	Use Case
UE	User Equipment
VM	Virtual Machine
VNF	Virtual Network Function
VNFD	VNF Descriptors
VNFFG	VNF Forwarding Graphs
WG	Working Group
WP	Work Package

1 Introduction

1.1 Motivation and scope clarification

SliceNet aims at defining a vertical-oriented, Quality of Experience (QoE)-driven network slicing framework that focuses on cognitive network management and control of end-to-end network slicing operation and slice-based (and slice-enabled) services in 5G networks. An effective and efficient orchestration, management and control of 5G network slices is crucial for delivering a wide plethora of services customized for heterogeneous vertical businesses with often diverging requirements for Quality of Service (QoS), QoE and Service Level Agreement (SLA) at large. As a further challenge, SliceNet also tackle network slicing in multi-domain and cross-provider environments, therefore requiring additional and dedicated composition logics (at orchestration, management and control levels) for slice providers to offer and deliver to verticals end-to-end slices while abstracting the details of different providers involved.

While network slicing is being recognised as a fundamental technology in the emerging softwarised and virtualised 5G mobile networks to enable independent end-to-end logical networks on a shared physical infrastructure, still verticals and slice consumers role and involvement in the runtime control and management of their slices is very limited. In this context, SliceNet aims at going beyond this current practice by targeting a high involvement of verticals and slice consumers in all the phases of the end-to-end slice lifecycle, from slice/service definition, to provisioning and runtime control and management. The main SliceNet idea refers to the “Vertical-in-the-Loop” concept, that can be considered as one of the main innovative principles proposed. With SliceNet, verticals and slice consumers are able to have access to their slices and have the possibility to play on top of them through a set of tailored control operations.

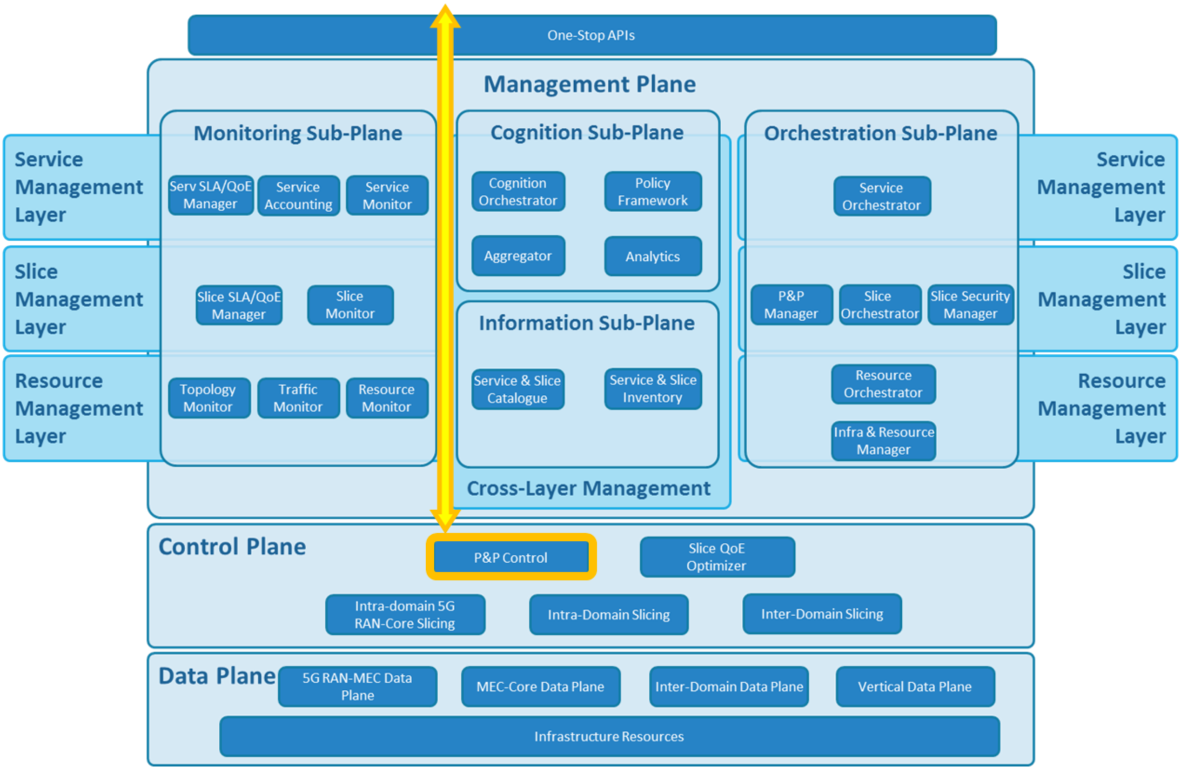


Figure 1 Plug & Play control within the SliceNet logical architecture

In this context, the Plug & Play control framework is the key enabler for the customization process of the slice exposed control. Dedicated vertical-tailored slice control instances are exposed and offered to verticals through the Plug & Play control framework, as isolated control environments where customized control functions and logics are plugged to build a truly tailored slice view. The idea is that each vertical may have different requirements and different expectations on how it would like to control, manage and monitor its slices, in terms of exposed (topological) view and runtime operations it could enforce. The Plug & Play control fulfils these requirements by providing a common framework able to expose customized slice control views and operations.

With respect to the SliceNet logical architecture [1], as depicted in Figure 1, the Plug & Play control framework is positioned within the Control Plane layer [2] as responsible for slices customization at runtime according to verticals requirements. However, the Plug & Play control is more than that. It can be considered as the overall enabler for vertical-tailored slice control exposure, and together with the One-Stop-API framework allows different verticals with very heterogeneous requirements and businesses to have runtime access to their slices following a common Plug & Play approach. Moreover, it is based on a very extensible and unified plugin-based design approach that allows additional Plug & Play vertical-tailored control functions to be developed and onboarded in the SliceNet system to augment the overall Plug & Play capabilities and portfolio, including Plug & Play features across different administrative domains and business roles boundaries.

1.2 Objectives of the document

This deliverable aims to achieve a set of technical objectives related with the Plug & Play control framework:

- Identify a set of core technical Plug & Play principles to act as ground for an innovative control exposure mechanism to be applied to network slicing
- Design an extensible and common Plug & Play slice exposure framework that allows any vertical and slice consumer to control their slices in a customized and tailored way
- Define interfaces and APIs of the Plug & Play control framework to ease its integration within the SliceNet platform
- Provide a Plug & Play control framework software prototype to be used (at least) in the context of the SmartGrid, SmartCity and eHealth use cases to offer tailored slice control instances to verticals

1.3 Document structure

This document is organized as follows:

- Section 2 provides an overview on current practices for control exposure offered by service providers and network operators towards their customer, with a gap analysis towards Plug & Play approach. Moreover, a brief insight on how other 5GPPP Phase 2 projects deal with verticals and slice exposure is included;
- Section 3 defines the main principles that have driven the design and implementation of the SliceNet Plug & Play control framework, as key enablers for an innovative solution aiming to customize and tailor how verticals and slice consumers accesses and control at runtime their slices;
- Section 4 is the core part of the document and provide the insights on the Plug & Play control architecture, with details on the microservice approach adopted and on workflows and mechanisms for the lifecycle of Plug & Play control instances. Moreover, the slice exposure information model is specified, as key driver of the Plug & Play approach, on top of which the Plug & Play APIs are also defined
- Section 5 provides a description of the Plug & Play control software prototype that has been implemented following the architecture and APIs definitions of Section 4. Here, brief installation and operation instructions are provided, together with the description of a preliminary integration and validation in the context of the SmartGrid use case.
- Section 6 provides some concluding remarks and highlights on future work.

2 Related work on exposure mechanisms

The main goal of this section is to describe the related activities about exposure mechanisms and identify the differentiating contribution from SliceNet. In detail, section 2.1 provides a practical exposure scenario from a service provider towards vertical industries (e.g. energy operators) that want to manage the connectivity of their devices. Thereafter, section 2.2 provides an overview on how other 5GPPP Phase 2 projects interacts with vertical actors, whereas section 2.3 addresses the key technical enablers for network exposure. Finally, in section 2.4, and before closing the section, a brief analysis about the missing steps from current management platforms towards a P&P approach are given.

2.1 Service Providers Exposure Mechanisms Overview

Although the service providers exposure mechanisms are typically non-disclosed information, or are, at least, very much dependent on the type of business relationship between the service provider and the vertical, we were able to obtain and share information about a specific exposure scenario from a service provider (Altice) within the SliceNet consortium. The used exposure platform – named Smart2M – is designed and developed in Altice Labs.

2.1.1 A practical example from Altice – Smart2M platform

In an age where society and technology combine in multiple and innovative ways, Altice Labs Smart2M platform provides telecommunications operators with a single, converged solution for managing connectivity across mobile devices in the field of machine-to-machine communications (M2M), translating into advantages sustained in a highly competitive market. It also provides to the Customer a Self-Care portal that provides means for intelligent and efficient management of M2M communications, anytime and anywhere, allowing efficiency gain, streamline processes, work in real-time and costs control.

The Smart2M solution, presented in Figure 2, has a modular and distributed architecture, ensuring functional and integration flexibility, as well as high levels of performance, scalability and fault tolerance.

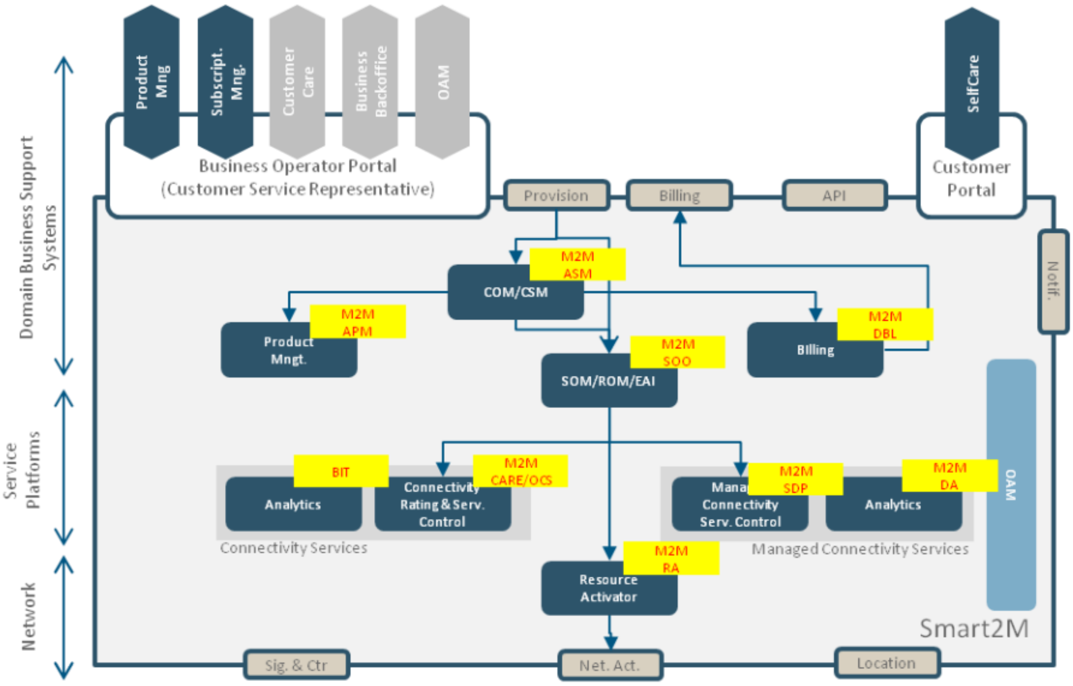


Figure 2 Smart2M high level architecture

From Figure 2 we can see two interfaces, one for the Service Provider (BOP – Business Operator Portal) and the other one for the Customer/Vertical (CP - Customer Portal). A high-level overview of the functionalities exposed to the vertical (through the Customer Portal) is described below.

2.1.1.1 Exposing a Self-Care Portal to the Vertical

The Self-Care Portal application presents a set of features, briefly illustrated in Figure 3, which provides to the customer the ability to:

- View on a dashboard area a set of indicators of activity related to the managed connectivity, providing each time a general perspective of the different managed layers;
- Managing the lifecycle of Resources / SIM cards with ability to run state-changing operations (ex.: activate, suspend), configuration update (add or remove communication services, tariff change, assign APN or IP address) or to order a SIM replacement, allowing the end user to work at any time on M2M connectivity, according to the operational and business needs;
- Consult information on the contracted fees and consumed amounts related to communications of SIM cards, allowing to continuous monitor and control communications consumption;
- Generate detailed communications reports (file) performed by one or more connectivity, thus allowing access to detailed information of communication sessions made by M2M devices;
- Alert configuration (enable or disable and define alerts thresholds) associated with consumption (ex.: consumption volume of shared account data) or other communication events (ex.: IMEI change alerts), allowing control costs and detection of abnormal situations (ex.: consumption above expected values, fraud, etc.);
- Enable actions: notifications to be sent by SMS and/or e-mail to a pre-defined recipient or to a group of recipients, set communications suspension on the occurrence of an alert, facilitating rapid decision-making in response to the alert;
- Manage the register of M2M equipment (M2M endpoints), with the potential of associating a set of specific data (identification, information equipment, installation address and geo-location information, technical contact and group to which it belongs) to one or more resources, facilitating efficient management of equipment park;
- Order a set of new resources (SIM cards);
- It allows to perform operations on a set of resources, Endpoints or IMEI Alerts in one operation, by uploading a file containing the pre-defined parameters;
- SMS console for sending SMS messages to resources and consultation of messages received by the resources to the platform.

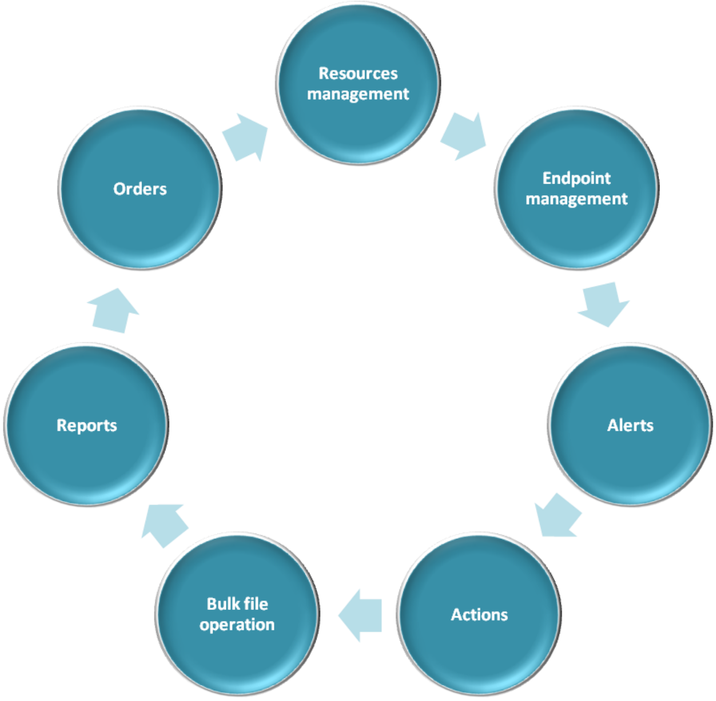
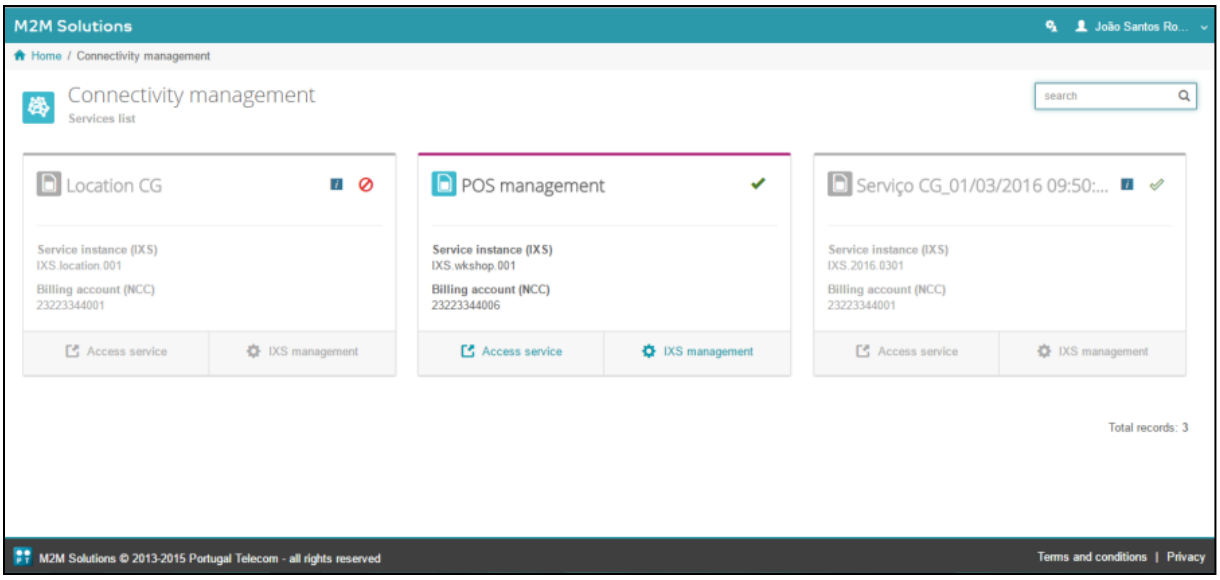


Figure 3 Self Care Portal features exposed to the Vertical

After valid authentication, the user accesses the portal services, depicted in Figure 4, in which all service instances and related actions to which the user is allowed to access are presented.



Connectivity management
Services list

Location CG
Service instance (IXS)
IXS.location.001
Billing account (NCC)
23223344001

POS management
Service instance (IXS)
IXS.wkshop.001
Billing account (NCC)
23223344006

Serviço CG_01/03/2016 09:50:...
Service instance (IXS)
IXS.2016.0301
Billing account (NCC)
23223344001

Total records: 3

M2M Solutions © 2013-2015 Portugal Telecom - all rights reserved

Figure 4: Self-Care Portal – Available Service Accounts

In Figure 4, in the Services list, three service instances are available. Each one of these service instances is characterized by:

- **Service account type**: presented together with iconography;
- **Service identification**: with a friendly name of the service (e.g. Location CG, POS Management, etc.);
- **Status**: describe the service instance status. Possible values are the following:
  - ✓ Pre-Active; ✔ Active and ⊘ Inactive.

Although relatively simple, the described exposure platform scenario shows that service providers are already tackling the evolution towards opening their assets to be managed by external entities, but still with limited exposure of runtime control routines.

2.2 R&D Projects exploiting the Network Exposition Principles

This section briefly introduces the most relevant ongoing 5GPPP Phase 2 projects related with exposure mechanisms and direct interactions with vertical actors.

2.2.1 5G-Transformer

The 5G-Transformer project [3] aims to transform today's rigid mobile transport networks into an SDN/NFV-based Mobile Transport and Computing Platform (5GT-MTP), that brings the network slicing paradigm into mobile transport networks by provisioning and managing slices tailored to the specific needs of vertical industries.

The system architecture is composed of three major components (see Figure 5):

- Vertical Slicer (5GT-VS), the entry point for the vertical requesting a service and it handles the association of these services with slices as well as network slice management
- Service Orchestrator (5GT-SO), responsible for end-to-end orchestration of services across multiple domains and for aggregating local and federated (i.e., from peer domains) resources and services and exposing them to the 5GT-VS in a unified way
- Mobile Transport and computing Platform (5GT-MTP), provides and manages the virtual and physical IT and network resources on which service components are eventually deployed. It also decides on the abstraction level offered to the 5GT-SO.

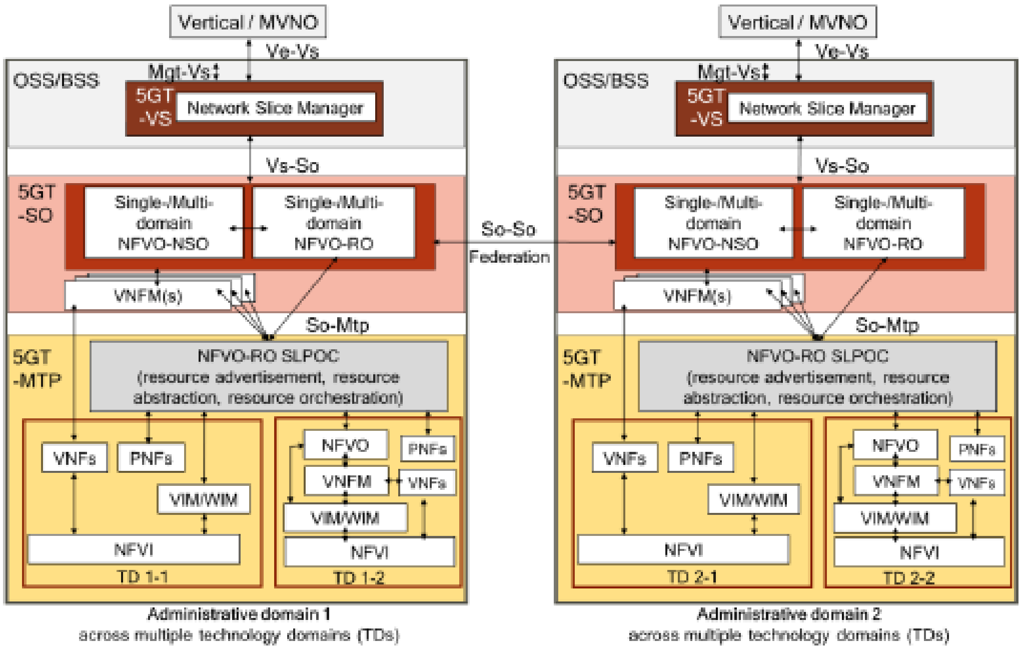

Vertical / MVNO
Ve-Vs
OSS/BSS
Mgt-Vs
5GT -VS
Network Slice Manager
Vs-So
5GT -SO
Single-/Multi-domain NFVO-NSO
Single-/Multi-domain NFVO-RO
VNFM(s)
So-Mtp
5GT -MTP
NFVO-RO SLPOC (resource advertisement, resource abstraction, resource orchestration)
VNFs
PNFs
VIM/WIM
NFVI
TD 1-1
NFVO
PNFs
VNFM
VNFs
VIM/WIM
NFVI
TD 1-2
So-So Federation
Administrative domain 1 across multiple technology domains (TDs)
Vertical / MVNO
Ve-Vs
OSS/BSS
Mgt-Vs
5GT -VS
Network Slice Manager
Vs-So
5GT -SO
Single-/Multi-domain NFVO-NSO
Single-/Multi-domain NFVO-RO
VNFM(s)
So-Mtp
5GT -MTP
NFVO-RO SLPOC (resource advertisement, resource abstraction, resource orchestration)
VNFs
PNFs
VIM/WIM
NFVI
TD 2-1
NFVO
PNFs
VNFM
VNFs
VIM/WIM
NFVI
TD 2-2
Administrative domain 2 across multiple technology domains (TDs)

Figure 5: 5G Transformer System Architecture

The 5GT-VS is the common entry point for all verticals into the 5G-Transformer system. It is part of the operating and business support systems (OSS/BSS) of the 5G-Transformer Service Provider (TSP). Vertical services are offered through a high-level interface at the 5GT-VS northbound that is designed to allow verticals to focus on the service logic and requirements, without caring on how they are eventually deployed at the resource level. In fact, the 5GT-VS offers a catalogue of vertical service blueprints, based on which the vertical service requests are generated by the vertical. The role of the

5GT-VS is to trigger the actions allowing the 5G-Transformer system to fulfill the requirements of a given incoming service request. The 5G-VS is the component of the system that is conscious of the business needs of the vertical, their SLA requirements, and how they are satisfied by mapping them to given slices.

Through this process, the 5GT-VS maps vertical service descriptions and instantiation parameters at the vertical application (VA) level into an NFV Network Services (existing or new) implementing the network slice. In turn, such NFV Network Services will be updated or created through a network service descriptor (NSD), which is a service graph composed of a set of virtual network functions (VNF) chained with each other, and the corresponding fine-grained instantiation parameters (e.g., deployment flavour) that are sent to the 5GT-SO.

2.2.2 MATILDA

The vision of MATILDA [4] is to design and implement a novel holistic 5G end-to-end services operational framework tackling the overall lifecycle of design, development and orchestration of 5G-ready applications and 5G network services over programmable infrastructure, following a unified programmability model and a set of control abstractions. MATILDA aims to devise and realize a radical shift in the development of software for 5G-ready applications, as well as virtual and physical network functions and network services, through the adoption of a unified programmability model, the definition of proper abstractions and the creation of an open development environment that may be used by application as well as network functions developers.

Generally speaking, appropriate abstractions of the underlying infrastructure and capabilities are desired to lower the barriers for creating 5G-ready applications that are able to satisfy business and user necessities. The purpose of the MATILDA project is to provide such abstractions by establishing a holistic framework that unifies the development, deployment and operation for this new kind of applications as well as automating most of those processes.

The MATILDA layers, along with the main artefacts and key technological concepts comprising the MATILDA framework per layer, are depicted in Figure 6.

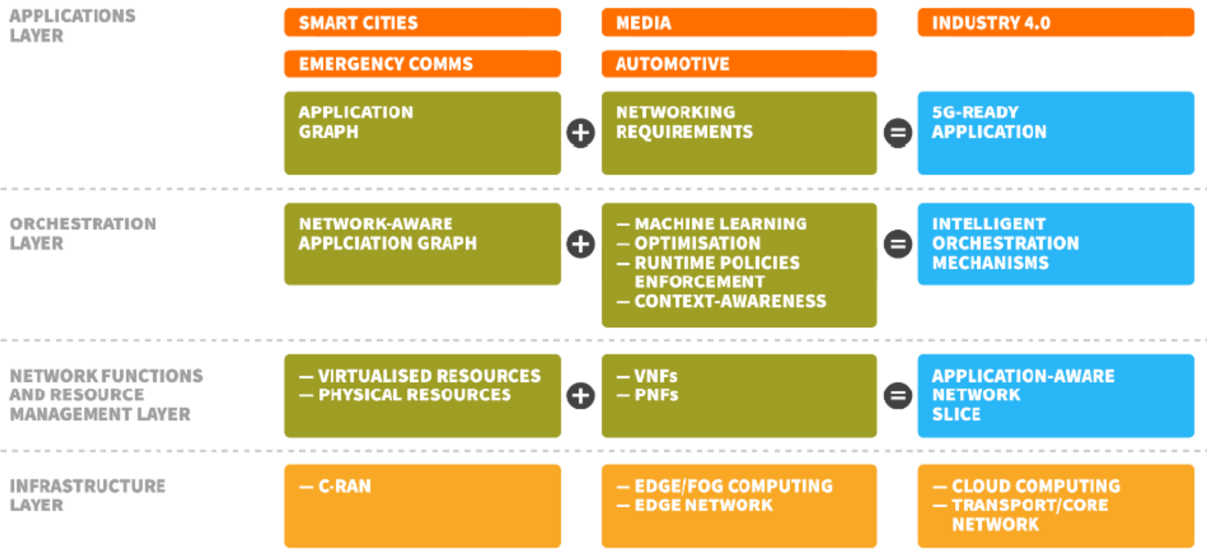


Figure 6: MATILDA High Level Architectural Approach

Specifically, to enable the creation of 5G-ready applications by third party entities, the Applications Layer allows vertical actors to define an application in the form of chainable application components, which result from the abstraction of network and virtual resources. Each of them adheres to a specific meta-model describing their capabilities, with the combination of them representing the application graph. Following this graph model, lower layers, such as the Orchestration Layer, are in charge of fulfilling the specific resource requirements that are needed to support the defined application. In this regard, the Application Layer follows a Service Development Kit (SDK) approach that hides the internal complexities of service and resource delivery from external application developers, allowing for a more agnostic application design (driven by verticals) and deployment.

2.2.3 5GCity

The 5GCity [5] project gives to the municipality the role of an infrastructure owner that can rent pools of virtualized resources to different (virtual) service providers. In this context, each provider may offer different kinds of services (media production, broadcast and content distribution, connectivity – e.g. virtual mobile operators, etc.) and, thus, may have its own requirements in terms of used technology, configuration and management of the contracted virtual infrastructure [6].

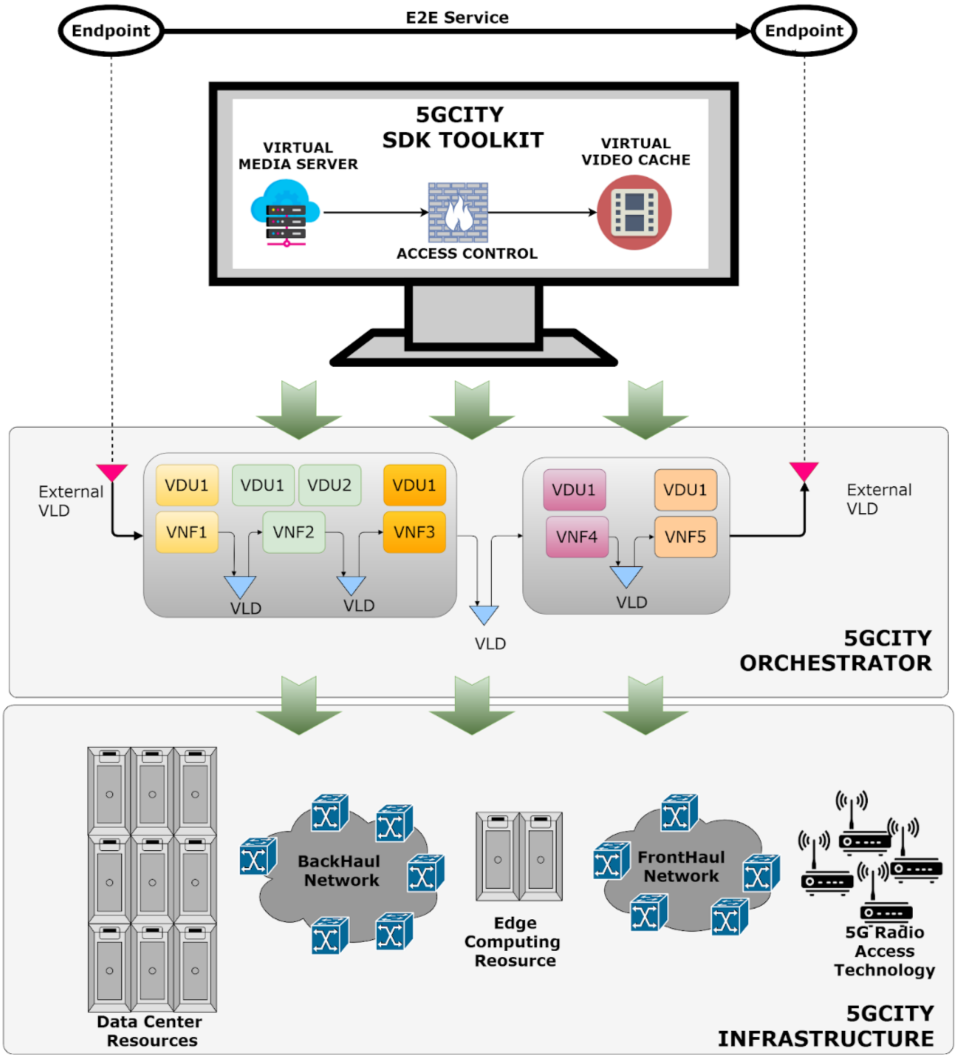


Figure 7 5GCity project architecture overview [6]

In light of the above, one of the goals of the 5GCity project is to leverage an SDK principle to open-up the virtualization advantages to third party vertical industries. To go beyond the existing SDK toolkits, which typically follow a network-centric approach where network services and VNFs can be defined and tested before they are instantiated through the orchestration plane, 5GCity targets at offering a service-centric approach. This consists of offering a set of tools to provide virtual operators with service design and deployment over the 5G network infrastructure.

Figure 7 depicts a simplified diagram of the 5GCity vision. There, the SDK resides on top of the orchestration plane and acts as the system entry point for (virtual) service providers' administration. Then, the orchestration plane is responsible for configuring the 5GCity network infrastructure according to the service definitions provided through the SDK.

The 5GCity SDK has three main components, namely a dashboard (GUI), an adaptation layer and a validation module. The first one provides a user-friendly way to define network services. The second one hides the low-level infrastructure details and translates the functional components and business requirements into an operational service to be virtualized and deployed over the infrastructure. The third one is responsible for validating the service designed by the service providers.

2.2.4 5G ESSENCE

The 5G ESSENCE project will take advantage of the Edge Cloud computing and Small Cell as a Service paradigms to provide a neutral host market, based on a highly flexible and scalable platform, aimed to offer new opportunities for infrastructure ownership, deployment, operation and amortisation[7].

From a technical perspective, 5G ESSENCE plans to achieve the benefits of Cloud-RAN but avoiding the fronthaul latency restrictions. This will be achieved by moving the Small Cell functions to an edge cloud environment based in a two-tier architecture. The first tier will provide low latency for services requiring it, and the second one will provide high processing power for computation-intensive services.

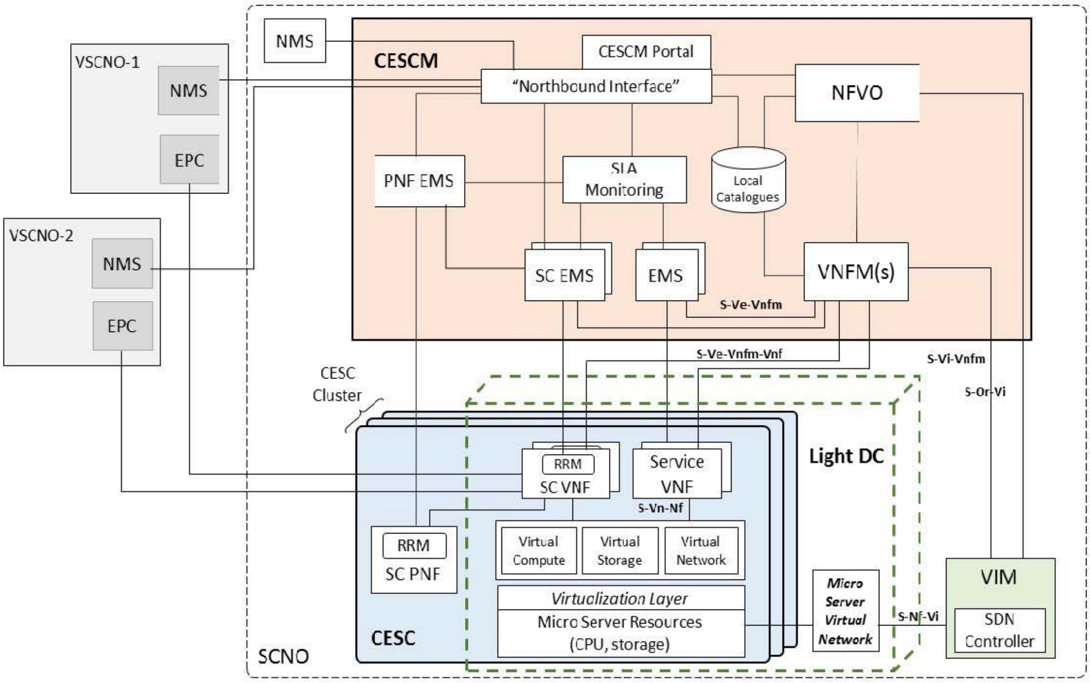


Figure 8 SESAME general architecture [9]

With regards to the network exposition, 5G ESSENCE targets at exposing a customisable interface to the customers. In particular, the project has as an objective to define the system and interfaces to provision a per-vertical customisable cloud-integrated multi-tenant small cells network and a programmable Radio Resource Management (RRM) controller. To achieve this, the 5G ESSENCE project will rely on the knowledge and software prototypes of the 5GPPP Phase 1 SESAME project [8]. Specifically, the Cloud-Enabled Small Cells (CESC) Manager will be the responsible entity for the network exposure [9]. In this context, the CESC Manager (CESCM) provides a portal (with a web GUI) that serves as the entry point for users (Figure 8). It provides monitoring, SLA and available services information, as well as infrastructure and service parameters configuration. The portal supports two roles, namely administrator and user. The former is for the Small Cell Network Operator (SCNO, i.e.

the infrastructure owner) and allows to configure the CESCM elements. The latter is for Virtual Small Cell Network Operators (VSCNOs), which are the tenants, and allows for retrieving (visual) monitoring information about the running services and the current SLAs, and the creation, instantiation and deletion of services. A northbound interface enables the communication between the portal clients (administrators and users) and the CESCM.

2.2.5 5G MEDIA

The 5G MEDIA project [10] aims to design and develop an integrated programmable service platform that will enable an advanced management environment for the design, development and operation of media applications over 5G networks. The general architecture of the platform comprises three main blocks: The Application/Service Development Kit (SDK) that provides access to media applications agile development tools, the Service Virtualization Platform (SVP) that hosts the network management and orchestration components (such as NFV Management and Orchestration and the cognition-based network optimization module), and the infrastructure layer that comprises the physical and virtual network infrastructures [11].

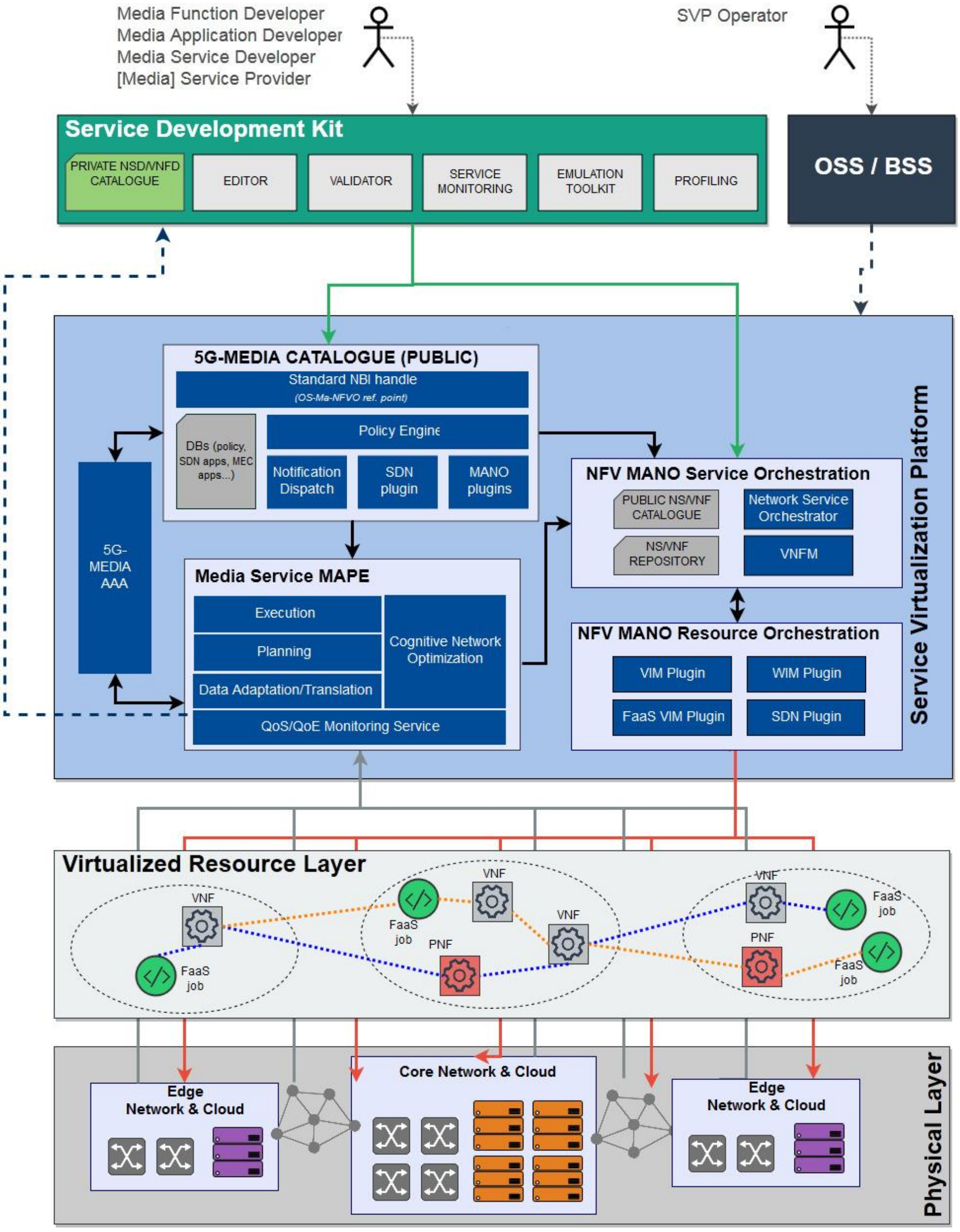


Figure 9 5G MEDIA general architecture [11]

The 5G MEDIA project plans develop a set of SDK tools that helps developers to implement media-related network applications, following the DevOps methodology, to be further deployed in the SVP. To do this, along with the classic elements of an SDK for edition, validation and packaging of the new services, the 5G MEDIA SDK will contain a set of tools to provide per-developer infrastructure exposition. In this regard, along with a platform catalogue, a private catalogue containing proprietary VNF and NS descriptors and images will be available for each developer. The SDK will also offer an emulation tool that will allow each developer to test the real functions over emulated network topologies. A monitoring tool will collect and visualize network monitoring data that will be exposed to the developer. In particular, this tool will offer both general VNF metrics (such as CPU, memory, etc.) and more specific traffic analysis ones. Finally, a profiling tool will allow developers to stage a service in a local SDK environment to overcome possible issues of a service before deploying it in the SVP for production.

2.2.6 R&D Related Projects Overview

As described in the previous subsections, most of the 5GPPP Phase 2 projects dealing with network slicing and provisioning of vertical-tailored offerings are able to deliver services and slices by processing and elaborating vertical requirements. A common choice is the usage of SDK approaches to allow slice consumers to specialize their services and slices at design phase, hiding the complexity, capabilities and resource granularities in the service provider and network operator infrastructures. Therefore, while it is clear that the interaction with vertical actors in the design phase is a common requirement and target of most of the proposed solutions, the intervention of verticals in the subscribed services and slices during runtime is still very limited. When it happens, these are mostly based on a pre-defined set of operations exposed to verticals, with a low degree of customization of both slice instance view (in terms of slice components) and runtime control APIs and logics exposed.

SliceNet aims at filling this gap and the Plug & Play control framework described in this document has the main objective to go beyond this current practice while offering to verticals dedicated per-slice control instances enabling a customized slice exposure that fulfil the vertical’s business logics and requirements for runtime control.

2.3 Technical Enablers for Network Exposure

A key use case for 5G network infrastructures relates to the delivery of specialized network services towards vertical customers, which encompass network, computing and service/function resources as foundation to provide services to third parties. To support them, a Network as a Service (NaaS) model is envisioned. Thus, a common network infrastructure can be leveraged by different verticals, partitioning the infrastructure in self-contained slices to support different network services. Such an approach not only requires an efficient way to slice the underlying physical infrastructure as well as means to virtualize its elements to later construct the desired virtual infrastructures, but also requires the means to expose control and customization capabilities of the slice towards third parties (e.g. the verticals) as to allow them to manipulate and access to the slice as they see fit.

Indeed, exposure of control and customization capabilities is one of the main requirements towards breaking the classic provider-consumer model, in which an entity (i.e. the provider) is focusing on offering a service (e.g. a slice) towards another entity (i.e. the consumer), assuming the responsibility for the control and management of the delivered service, while leaving no possibility of control or customization by part of the consumer. The emergence of new services and business models, as well as the need to boost innovation and competitiveness, is raising the necessity to open control and customization capabilities of the underlying physical infrastructure towards consumers. This allows consumers (e.g. vertical actors) to be engaged in the process of controlling and managing the requested service, either by reconfiguring its characteristics or by deploying custom functions to complement the base service and tailor it to their needs. In this way, consumers are no longer passive actors in the whole process of service delivery but participate actively in the process of managing and optimizing it.

Software Defined Networking (SDN) and Network Function Virtualization (NFV) can be considered as key technological enablers for control exposure as well as customization of network slices. On the one hand, SDN allows to expose control capabilities of the abstracted underlying physical network while on the other hand NFV enables verticals to deploy (virtualized) functions and appliances customized to their needs. The following subsections elaborate about these two paradigms, and how they can enable exposing network resources as well as their control and customization towards third-party users.

2.3.1 Network Programmability – SDN

The main rational behind the SDN paradigm is the decoupling of data and control plane functions [12]. Following such principle, data plane elements become simple forwarding elements while all the control logic is removed from them and placed onto a common centralized control layer (i.e. the SDN controller), which is the responsible for the control and configuration of the underlying physical network substrate. Such a decoupling facilitates the creation of independent logical abstractions of the network hardware elements and, in turn, allows for a more efficient way of slice creation, which was not possible, or more complicated, in legacy control architectures in which data and control plane functions were more tightly coupled (e.g. Generalize Multi-Protocol Label Switching (GMPLS)-based networks).

More specifically, SDN contributes towards facilitating the abstraction of the physical network complexity and expose its control capabilities to third party entities or applications. SDN standardizes the way in which data plane elements can be configured, for example, employing the same configuration protocol (e.g. OpenFlow [13], NETCONF [14]), as well as the exposure of their capabilities. By building up information models that describe macroscopically the hardware capabilities, it becomes possible for applications residing in the control layer to construct software (i.e. abstracted) representations of them, providing different views of the hardware element to fit heterogeneous requirements. These abstract elements are then bound to the sub-set of functionalities of the underlying physical elements and may expose their control capabilities through standard interfaces.

In addition to that, data plane capabilities may be capitalized by means of specific SDN applications that may run within the control layer. The rationale behind SDN applications is to provide high level complex operations (e.g. create an end-to-end connection, provide re-routing capabilities) while hiding the internal configurations and considerations required to achieve them. Thus, by means of simple and standard APIs, it is possible to expose simplified configuration of network services on top of the abstracted physical network, achieving programmatic control of the network by external entities (e.g. verticals) or applications.

2.3.2 Network Virtualization – NFV

The NFV paradigm aims to leverage on standard IT virtualization capabilities to overcome limitations imposed by standard hardware-based network deployments [15]. Traditionally, to support new functionalities and services, it is required to deploy new hardware (or upgrade existing one) that implements the desired new configurations to support them. This process not only is quite costly but also has the limitation that physical hardware may have a relatively short lifespan in terms of utility. NFV objective is to overcome these limitations by encapsulating network functions in the form of software appliances deployed in standard computing servers.

These software appliances, identified as Virtual Network Functions (VNFs), allow for a more flexible infrastructure deployment tailored to the needs of the service to be supported on top. In combination with SDN, to provide the connectivity between VNF instances deployed in different

Points of Presence (PoPs), NFV is the foundation for network slicing and vertical-tailored customization, with VNFs ad Network Services providing the specific flavour to accommodate service needs.

Since network functions are deployed as virtualized elements (e.g. Virtual Machines, containers) encapsulating the processes implementing them, it becomes possible to influence the behaviour of the network by parameterizing the configuration of the network functions. Indeed, software implementations of the NFV Management and Orchestration (MANO) architecture for management of VNF-based end-to-end services, offer the capability to expose VNF configurations toward service consumers, such as verticals, in the form of primitive functions to manage the lifecycle of VNF instances as well as their configuration. This opens up the possibility to expose to verticals the possibility to on-board and deploy their own custom VNF instances to enrich the slice in which they are operating. Thus, flexible and extensible instantiation of vertical functions is achieved by design, leveraging on the foundations of the NFV paradigm.

2.4 Current Management Platforms Evolution Towards a P&P Approach – a Gap Analysis

Telecommunications networks were designed and implemented to provide the various services without customers and verticals being able to intervene directly in their infrastructure. Service Level Agreements (SLAs) were established between service providers and customers and compliance with these SLAs was verified. Although these relationships are essential, the next generation telecommunications networks and service provider platforms are being designed so that customers and verticals can play a more active role. This role translates into controlled access to the service providers' infrastructures, for example instantiating or applying customized configurations to (virtualized) network functions, in order to optimize, update or deploy new innovative services in a flexible and agile way.

As already mentioned, there are already some approaches from service providers to this new interaction paradigm (for example the Smart2M case from Altice/Altice Labs mentioned in section 2.1.1), but they still do not exploit all the capabilities that are intended to be offered in these service provider platforms.

In order to achieve this objective the vertical requires two main interactions with the service provider:

- actuation on the network slice in order to control it (e.g., instantiation, configuration and reconfiguration of new VNFs, lifecycle management of slices, etc.);
- access to monitoring information related with the provisioned network slice (e.g., alarms, performance indicators, etc.).

The evolution of service providers operations shall accommodate these requirements. For example, dedicated interfaces and abstraction layers enabling the execution of these actuation and sensing interactions in runtime are paramount. For this purpose, service providers shall design and provide these features, while network operators should equip their networks with the necessary exposure features (i.e. leveraging on SDN and NFV technologies) so that the above two functionalities can be actually implemented. This will facilitate the required flexibility and agility for the vertical to optimize and also innovate his services.

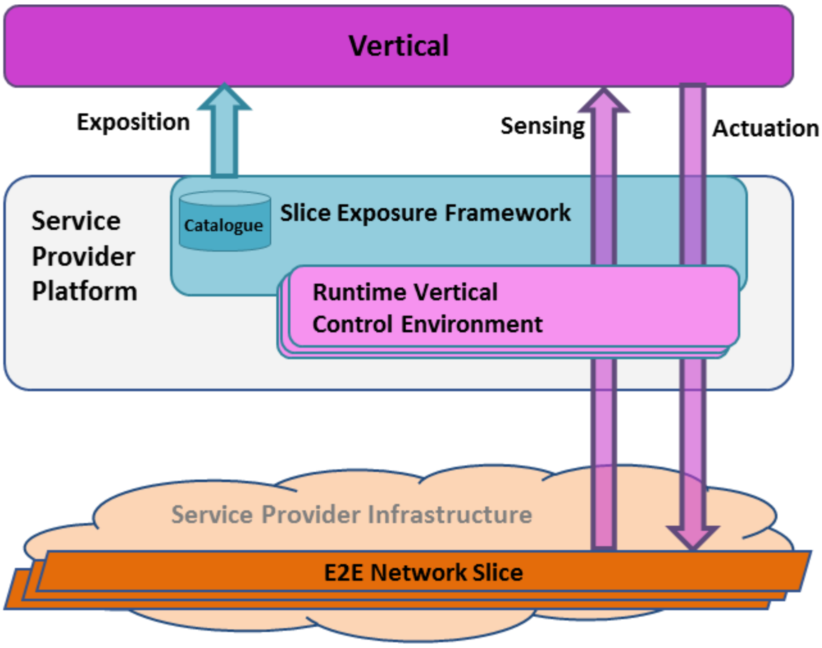

Vertical
Exposition
Sensing
Actuation
Service Provider Platform
Catalogue
Slice Exposure Framework
Runtime Vertical Control Environment
Service Provider Infrastructure
E2E Network Slice

Figure 10 A new exposure approach – the Vertical in the loop

Figure 10 shows a new service exposure paradigm to satisfy the above approach, where a comprehensive Slice Exposure Framework is able to address verticals requirements during the design phase, basically exposing slicing capabilities of the service provider and following what most of current solutions already offer. In addition, this exposure framework is also augmented with novel Runtime Vertical Control Environments, possibly dedicated per-slice, as a key differentiation feature to expose highly customized and vertical-tailored slice sensing and actuation to verticals.

SliceNet targets to offer this new slice exposure paradigm. The Plug & Play control framework (mapping to the Runtime Vertical Control Environments in Figure 10) described in the next sections aims at fulfil the gaps of current service provider and network operator management platforms towards fully customizable and vertical-tailored runtime control environments.

3 SliceNet Plug & Play principles

The Plug & Play control framework is designed and implemented around four main innovative principles, which are detailed in the following sections.

3.1 Vertical-In-The-Loop and exposed runtime control

The "Vertical-In-The-Loop" approach envisages a truly customized runtime control, management and operation of end-to-end slice instances in support of vertical-tailored services. In this context, the SliceNet Plug & Play control is among the key enablers of the "Vertical-In-The-Loop" runtime approach, as it provides an innovative combination of customized control functions, APIs and tools to enable verticals and slice consumers at large to even plug their own control logics and functions while specializing their slices according to their needs. This at the end offers significantly enhanced degree of flexibility for tailored services to end users.

The SliceNet Plug & Play control is designed to provide a new flavour of customization to end-to-end slice instances, from two orthogonal perspectives. First, from a slice consumer point of view, it allows verticals to drive (at least a restricted set of) the runtime control of their slices and services, where applicable and required offering the possibility to dynamically plug custom control and monitoring functions, thus enabling the deployment of specific 5G services in a truly customisable, dynamic and scalable way. On the other hand, from a slice provider perspective, the Plug & Play control enables the activation of all those specific per-slice control and management (including monitoring) functions needed to accommodate the vertical requirements, in terms of SLA, network functions composition, performance and QoE.

In summary, the "Vertical-In-The-Loop" approach is based on the proper grouping and mapping of management and control functionalities available in the context of the relevant SliceNet platform planes. This process can be considered as the definition of a concrete management workflow that serves a particular vertical oriented purpose. The purpose is expressed in the context of the slice control exposure as a northbound tailored operation that is addressing requirements defined by the verticals. The Plug & Play can support this approach as the workflow implementation can be prepared and instantiated on demand as well as exposed to the interaction interface provided to the vertical.

3.2 Customized control exposure and slice view

The SliceNet Plug & Play control framework aims at offering verticals an isolated control environment, including when required and applicable also management and monitoring features, specific per slice instance that can be activated on-demand upon new slice instances provisioning. The idea is that each Plug & Play control instance accesses a limited set of slice control and management primitives provided by the SliceNet platform, abstracting their logics and information models, according to the slice control exposure requirements specified by the slice consumer and agreed with the slice provider. This aims at offering verticals and slice consumers at large a tailored level of slice control exposure, that meets the consumer requirements and satisfies the provider confidentiality policies.

As a means to guarantee isolation across slice instances owned by different verticals (or slice consumers), each Plug & Play control instance insists and has access to those physical and virtualized resources and network functions (e.g. for configuration purposes), or monitoring information and metrics, owned and used by the given slice instance for which it is activated. As part of the tailored control exposure, the Plug & Play also offers the possibility to plug customized control (and management, including monitoring) functions allowing verticals and slice consumers to further specialize their slices and services.

Following the SliceNet control plane architecture principles defined in deliverable D2.3 [2], the Plug & Play provides a further control abstraction that aims to expose a simplified view of slice instances to verticals and slice consumers. On the one hand, this allows to specializing the slice control exposure level while aligning and complying with heterogeneous vertical logic and needs. On the other hand, it allows hiding and further abstracting the slice technology agnostic APIs and capabilities offered by the SliceNet control plane, fulfilling the agreements between consumers and providers in terms of control exposure.

As a key enabler towards this approach, the Plug & Play is built around a generic, common and technology-agnostic slice exposure information model, which for each slice is specialized according to vertical needs and agreed control exposure level. This Plug & Play slice exposure model (fully detailed in section 4.3) is implemented by an abstract slice view in the form of a generalized topology graph that allows customization of each element in the graph through the assignment of a tailored slice-context. In addition, each element in the graph is also augmented with the set of control operations (including management and monitoring when applicable) exposed. And as a more attractive and innovative aspect, this customization process is fully automated, not bound to pre-defined control endpoints, and open to dynamic updates.

3.3 Microservices and cloud native approach

The SliceNet Plug & Play control plane follows a microservices approach, targeting cloud-native setups. In particular, microservices is an architectural approach for developing an application as a collection of small services, each implementing self-contained capabilities, running its own process and communicating via well-defined APIs and messaging channels. Microservices can be deployed, upgraded, scaled, and restarted independently in the application, typically as part of an automated system, thus enabling runtime updates and upgrades without affecting overall offered services.

Following this principle, the SliceNet Plug & Play control plane is designed as a composition of microservices each providing a specific logic, from customized slice instance view, to dedicated plugins microservices responsible for the abstraction of SliceNet control and management plane primitives. In addition, this approach allows dynamically plugging into each Plug & Play instance additional customized control plane functions in the form of new microservices to further tailor the runtime control and management (including monitoring) of slice instances.

This Plug & Play application model is also fully aligned with the cloud-native approach, which aims at building and running applications exploiting the advantages of the cloud computing delivery model, thus caring more about how applications are created and deployed, rather than where. In other words, developing an application following the microservices architecture makes it automatically cloud-native. In this context, the Plug & Play aims at building and operating cloud-native applications and services that automates and integrates the concepts of microservices and containers. Indeed, containers offer a cloud-native way to deploy microservices based applications, as a more efficient, light and reactive approach compared to Virtual Machines (VMs) based silo deployments.

3.4 Flexibility and extensibility by design

The Plug & Play control framework is conceived to accommodate heterogeneous slice control exposure paradigms to fit different types of verticals. This is achieved by decoupling what is offered to verticals from the control capabilities provided the SliceNet platform by means of highly customisable Plug & Play microservices. In particular, Plug & Play control instances are composed by a variable collection of microservices, providing control capabilities abstraction and slice tailored view that can be flexibly adapted to vertical requirements and expectations. This is driven by dedicated plugins and control functions microservices that are conceived to be activated on-demand in the context of each Plug & Play control instance to enable respectively customized access to

SliceNet control, management and monitoring primitives on the one hand, and enhanced vertical-tailored complex control logics on top of SliceNet platform capabilities on the other.

In this context, the Plug & Play microservices approach itself natively offers dynamicity and flexibility in the way the slices are exposed to verticals, in terms of control capabilities offered and runtime operations allowed. This is because the plugins and control functions microservices enabling vertical-tailored slice control exposure can be easily orchestrated as containerized applications with very dynamic lifecycle procedures (i.e. to start, scale, stop them) that allow expanding, enhancing and updating the control capabilities offered to verticals in a very flexible and transparent way at runtime.

4 Plug & Play control plane for sliced networks

SliceNet aims at providing truly customized runtime control, management and operation of end-to-end slice instances while offering vertical-tailored services. The SliceNet Plug & Play (from now on referred as P&P) control framework is one of the key enablers of slice customization, as it provides a novel combination of tailored control functions, APIs and tools that are offered to verticals to let them plug their own control logics and specialize their slice instances according to their needs.

In practice, the P&P control plane provides an innovative control environment, dedicated per slice, which offers to the verticals, and in general to slice consumers, significantly enhanced degree of flexibility for deploying services to end customers. It exposes direct control of slice instances and provides a new flavour of customization. Deliverable D2.3 [2] describes the overall P&P approach and high-level functional decomposition, while this document, and in particular this section, intends to provide the full P&P control plane specification, including information models, internal architecture, deployment models, interfaces, software prototype, and detailing how the principles described in section 3 are translated into concrete P&P features and functionalities, thus evolving what was presented in D2.3.

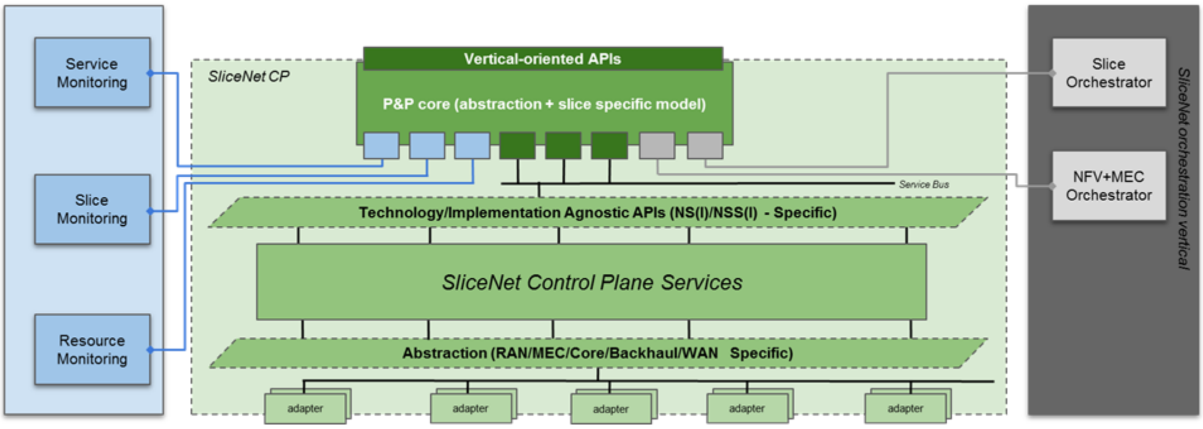

Service Monitoring
Slice Monitoring
Resource Monitoring
SliceNet CP
Vertical-oriented APIs
P&P core (abstraction + slice specific model)
Service Bus
Technology/Implementation Agnostic APIs (NS(I)/NSS(I) - Specific)
SliceNet Control Plane Services
Abstraction (RAN/MEC/Core/Backhaul/WAN Specific)
adapter
adapter
adapter
adapter
adapter
Slice Orchestrator
NFV+MEC Orchestrator
SliceNet orchestration vertical

Figure 11: SliceNet P&P main interactions with other SliceNet components [2]

Figure 11 shows how the P&P integrates within the overall SliceNet framework, in line with the D2.3 specifications, and highlights how each customized P&P control instance interacts with external monitoring, slice control plane, and management/orchestration components. The main idea is that the P&P provides, for each slice instance, a vertical-tailored abstraction of the slice (including its view and control exposure) on top of the SliceNet control plane technology and implementation agnostic APIs. The P&P is therefore the place for slice customization, where tailored control functions can be plugged to specialize the behaviour and performance of slice instances. This happens through the interaction of each P&P control instance with a set of SliceNet control and management APIs (through dedicated and specialized plugins) exposed by:

- *service, slice and resource monitoring services*: to retrieve KPIs, metrics and status information at different granularities (i.e. resource, slice and service) and when required to further elaborate.
- *SliceNet control plane*: to enforce vertical-tailored runtime slices configurations leveraging on the slice technology and implementation agnostic APIs, following the SliceNet control plane Service Based Approach (SBA) thus accessing the service bus and discovering the available APIs and services.
- *Slice and NFV/MEC orchestration*: to access slice lifecycle management primitives, and if and when required NFV/MEC orchestration ones, exposed by the SliceNet management and orchestration components, and enabling verticals to directly control deployment, scaling or termination of customized vertical MEC Applications or VNFs, possibly at specified locations.

Deliverable D2.3 also presented an initial high-level function decomposition of P&P control instances, which is reported for sake of completeness of this document in Figure 12. It follows a layered approach composed by three main high-level modules, each aimed at providing a high degree of flexibility and configurability in creating customized views of slice instances for verticals and slice consumers. The three main P&P modules are briefly described as follows:

- *Plugins and control functions*: this layer includes the set of pluggable modules providing the customized control functions in the context of each slice. These control functions can be mainly of two categories: i) pluggable control functions implementing a vertical-tailored logic (either at control, management or monitoring level), ii) plugins providing the required adaptation between the P&P generic slice exposure model and the SliceNet framework primitives, acting therefore as hooks for the SliceNet control, management and monitoring APIs
- *Abstraction and slice-specific model*: this layer provides the customization of the generic exposure slice information model (that provides an abstraction of the information model specified in D2.4 [16]) into a tailored vertical view of the reference slice instance. It offers an abstract and vertical friendly view, on top of which it adds the logic for the control operations that can be consumed by verticals at runtime over their slices. This is done by means of a dynamic and automatic mapping of operations issued from the vertical oriented APIs into specific actions to be enforced or performed through the pluggable control functions.
- *Vertical-oriented APIs*: this layer implements the set of tailored control and management APIs that are exposed to the vertical. It basically offers the runtime control operations over the slice-specific model and view, following the slice control exposure capabilities agreed by slice consumer and slice provider. The verticals access these APIs through the One-Stop-API framework, which somehow embeds the vertical oriented APIs providing all the necessary authentication and user management logics required for a secure access to third parties.

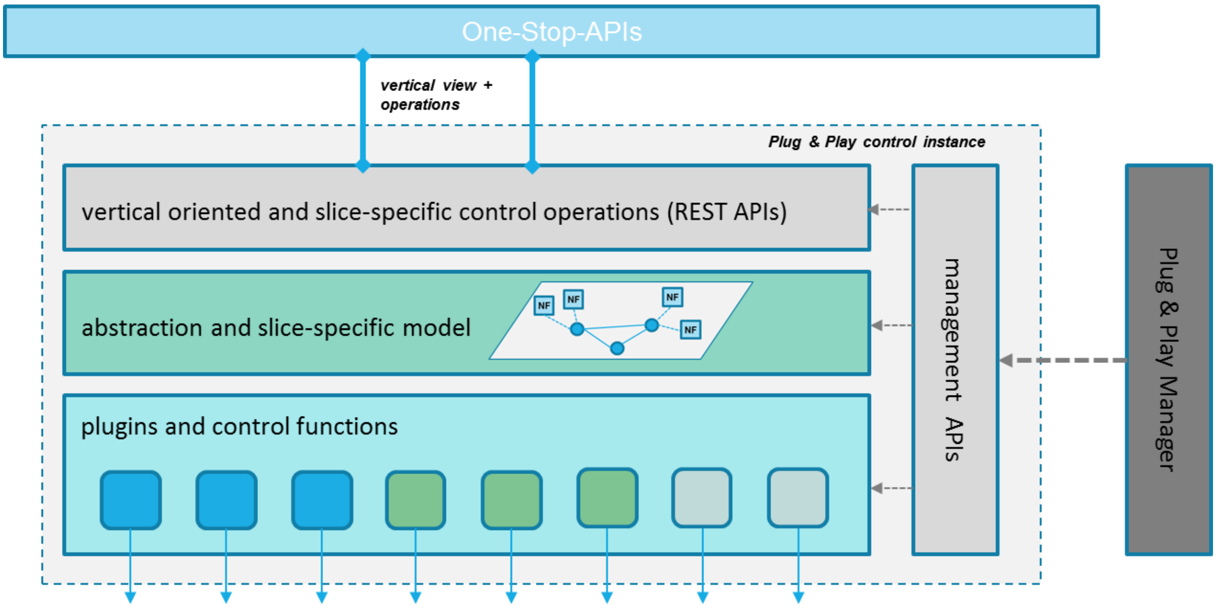

One-Stop-APIs
vertical view + operations
Plug & Play control instance
vertical oriented and slice-specific control operations (REST APIs)
abstraction and slice-specific model
NF
NF
NF
NF
plugins and control functions
management APIs
Plug & Play Manager

Figure 12 Plug & Play control instance high level functional decomposition [2]

4.1 High level architecture

The P&P control plane is considered as the SliceNet enabling framework for slice customization and vertical-tailored control exposure. The slice specialization paradigm offered by the P&P covers two fundamental aspects:

i. how the slice is exposed, i.e. how it is presented to verticals and slice consumers in terms of components, network and service functions, topology;
ii. how the slice and its components can be controlled and managed at runtime, i.e. which tailored and customized runtime operations can be applied by vertical and slice consumers.

One of the main and key features enabling this two-fold flexible control exposure is the P&P microservices approach. The layered high level architecture shown in Figure 12 is practically implemented by each P&P instance as a collection of self-contained microservices, as depicted in Figure 13. The microservices-based architecture enables a truly dynamic and flexible approach, as P&P components can be activated or updated at runtime without disrupting the overall P&P control instance integrity and continuity, in particular when selecting proper technologies for containerization and orchestration of microservices (e.g. Docker [17] and Kubernetes [18] as described in section 5).

The following sub-sections provide a brief insight on the four different types of P&P microservices represented in Figure 13 and composing the P&P control instances. Even if their logic and purpose are substantially different, the plugin and control function microservices are grouped together to describe the common internal functions and features they have to implement to be compliant with the SliceNet P&P paradigm and approach.

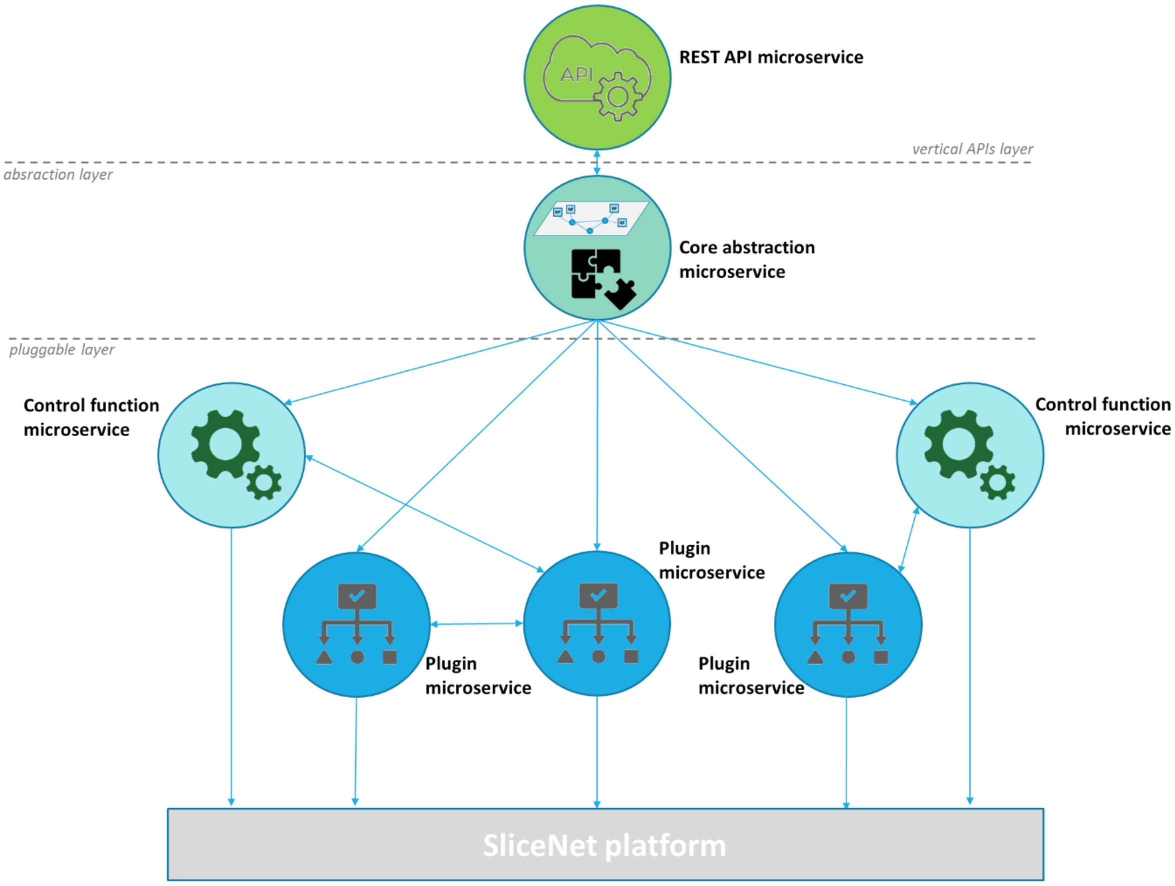


Figure 13 Plug & Play control plane microservices architecture

4.1.1 Plug & Play microservices

Each P&P control instance is the composition and integration of a set of microservices, each implementing a well-known and defined function. As shown in the Figure 13, different types of microservices can be deployed and activated as part of a P&P instance. It is relevant to highlight that Figure 13 depicts a reference deployment of a P&P control instance that has to be considered as a reference example. The exact composition of the P&P microservices strictly depends on the specific customization of the slice control exposure that the P&P instance itself has to offer. However, each P&P control instance needs to be composed by at least: i) the vertical tailored REST API microservice, ii) the core abstraction microservice, iii) at least one control function or plugin microservice that provides access to SliceNet platform data stores or control and management APIs.

4.1.1.1 Plugin and control function microservices

The plugin and control function microservices map to the lower layer of Figure 12, being the one providing the adaptation and hooks towards the SliceNet control, management and monitoring functionalities. Both these types of microservices follow a similar internal functional architecture split, still with substantial differences in their logic and purpose. Figure 14 shows this common functional approach for the P&P plugins and control functions.

The plugin microservices provide hooks to the SliceNet platform APIs, data stores and logics, and expose the related operations to the core abstraction and control function microservices through dedicated plugin-specifc REST APIs. Therefore, they can be considered simpler microservices in terms of internal logic as they just provide access to well-defined SliceNet control, management and orchestration primitives, including monitoring data stores, while applying translation and light abstraction only when required. Each plugin implements a specific logic to wire the APIs it accesses in the SliceNet platform (through the plugin-specific southbound driver) with those exposed to the P&P core abstraction microservice. Different types of plugin microservices exist according to the nature of the functionality accessed in the SliceNet platform, that can be related to monitoring data, control plane, slice and service orchestration.

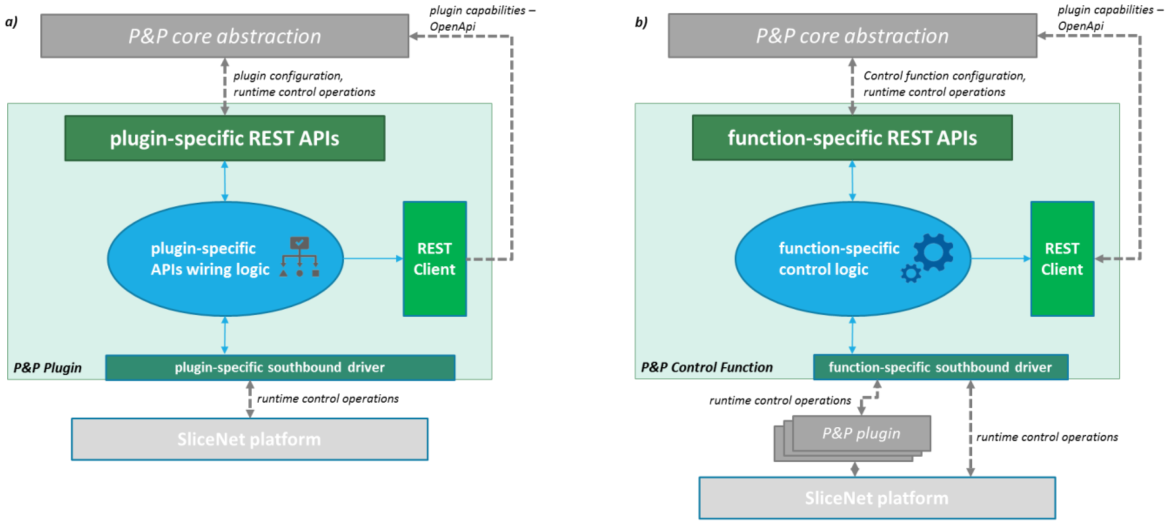

a)
P&P core abstraction
plugin capabilities – OpenApi
plugin configuration, runtime control operations
plugin-specific REST APIs
plugin-specific APIs wiring logic
REST Client
P&P Plugin
plugin-specific southbound driver
runtime control operations
SliceNet platform
b)
P&P core abstraction
plugin capabilities – OpenApi
Control function configuration, runtime control operations
function-specific REST APIs
function-specific control logic
REST Client
P&P Control Function
function-specific southbound driver
runtime control operations
P&P plugin
runtime control operations
SliceNet platform

Figure 14 P&P plugin (a) and control function (b) common functional split

On the other hand, the control function microservices are self-contained functions implementing a vertical-tailored control logic, thus leveraging and coordinating APIs and data collected from other P&P plugins (in line with the microservice approach) or directly from the SliceNet platform, while exposing to the core abstraction microservice (and thus to verticals) new APIs and logics not natively offered by the SliceNet system. Therefore, each P&P control function allows further specializing the dedicated slice control environment offered to verticals in order to meet the required level of control exposure agreed. Some of the QoE sensors and actuators envisaged in SliceNet can be also deployed as P&P control functions and thus provide specialized QoE optimization features for a given vertical slice (or set of slices).

For both types of microservices, as depicted in Figure 14, a REST client allows registering the P&P plugin or control functions to the core abstraction microservice. This registration process allows to advertising the control primitives exposed by the plugin or control function in the form of an

OpenAPI specification [19]. This operation is performed on the interface described in section 4.3.3 and enables dynamicity and flexibility in the lifecycle of the P&P instances and the related exposed control features.

It is relevant to highlight that P&P plugins and control functions are expected to be offered by the slice provider entity (e.g. a network service provider) and thus be already available in the SliceNet platform, as part of the overall P&P capabilities to be exposed and offered to verticals. For this, a P&P catalogue is envisaged to be part of the P&P Manager component in the SliceNet management layer [16], as further highlighted in section 4.2. For each P&P control instance, the logic for the selection (and deployment) of the proper set of plugins and control functions microservices is out of the scope of the given P&P control instance and reside into the SliceNet management layer, in particular in the P&P Manager component that is in charge of managing the lifecycle of all the P&P control instances.

The P&P plugins and control functions are therefore key components for offering a customized slice control exposure. The design (and the microservices approach) chosen enable an extreme flexibility and extensibility of the whole P&P control framework allowing to develop new plugins and control functions at any time to fulfil specific new vertical requirements, e.g. related to tailored slice lifecycle management logics, exposure of particular slice monitoring data and aggregated metrics, control of virtual appliances (e.g. VNFs, MEC applications) provided by the vertical.

However, an initial set of P&P plugins and control functions has been implemented following the design specified above, as reported in the software prototype description in section 5.3. These have been developed in the context of a preliminary integration and deployment of the SmartGrid use case Phase 1 [20]. The plan is to further elaborate in the context of WP8 activities the P&P requirements initially identified and reported in D2.1 [21], and augment the P&P control framework with more vertical-tailored control functions and plugins.

4.1.1.2 Core abstraction microservice

The core abstraction microservice is the core engine of each P&P control instance, and it implements the abstraction and slice specific model layer of Figure 12. It is the main enabler of the slice control exposure customization, in terms of both vertical-tailored slice view (i.e. network and service functions components) and offered runtime control (and management, when applicable) operations. In particular, the core abstraction microservice translates and abstracts the specific APIs, logics and data offered by the plugin and control function microservices into vertical-tailored operations. To do that, it manages in a fully dynamic and flexible way two different slice control exposure views: an internal one that keeps the wiring with the control capabilities offered by plugins and control functions, and an external one that is offered to the vertical-tailored REST API microservice and models the exposed control operations only.

Therefore, the P&P control is built around a technology-agnostic slice exposure model that allows to represent a given slice instance as a collection of slice elements organized in a topology, augmented with a set of control capabilities, attributes, operations and primitives exposed on top of it. The differentiation and separation between the internal slice exposure model and the one directly exposed to verticals is required to hide the internal details and logics of plugins and control functions, as well as of the SliceNet platform capabilities. For each P&P control instance, the internal slice control exposure model is instantiated in a customized slice view that the core abstraction microservice uses to offer a tailored slice control to the vertical. More details about the P&P control exposure information models are provided in 4.3.

Figure 15 presents the functional decomposition of the P&P core abstraction microservice, that is responsible to: i) handle plugins and control functions registration, ii) maintain the vertical-tailored slice control exposure information consistent, iii) map runtime control operations from the vertical into control operations in the plugins and control functions.

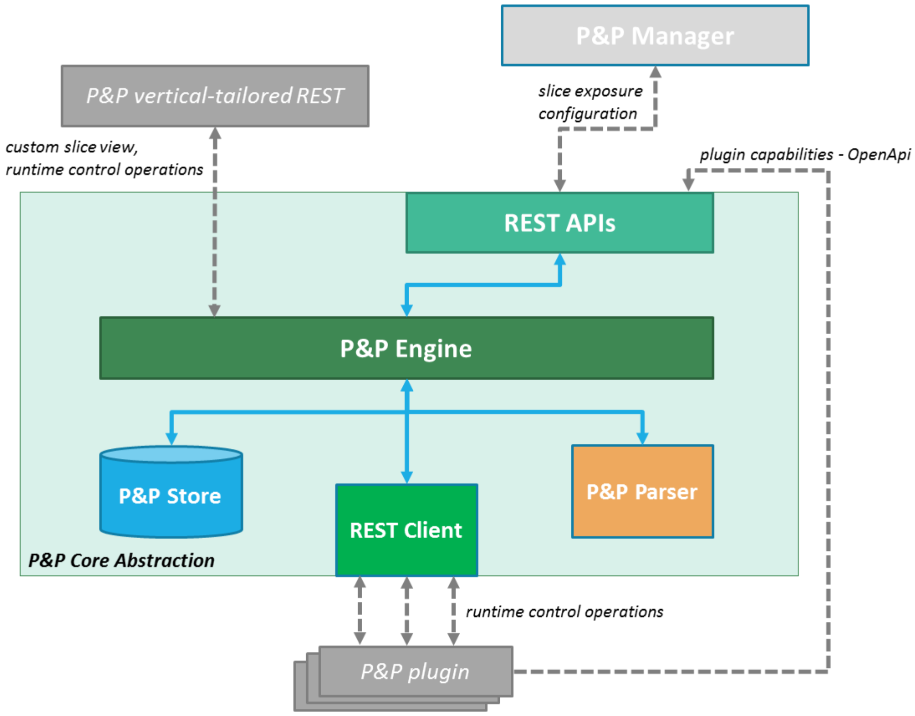

P&P Manager
P&P vertical-tailored REST
slice exposure configuration
custom slice view, runtime control operations
plugin capabilities - OpenApi
REST APIs
P&P Engine
P&P Store
REST Client
P&P Parser
P&P Core Abstraction
runtime control operations
P&P plugin

Figure 15 P&P core abstraction microservice functional split

The P&P Engine is the core component within the core abstraction microservice, as it coordinates all the slice control exposure configurations and customizations, as well as the registration of P&P control functions and plugins and the runtime control operations coming from the P&P vertical-tailored REST API microservice. In particular the P&P Engine validates the customized slice control exposure configurations (as instances of the internal slice exposure model) coming from the P&P Manager (through the REST APIs in Figure 15) and maps the vertical-tailored slice view and control operations expressed in such configurations to the available plugins and control functions capabilities.

To enable this mapping within the P&P Engine, each plugin and control function microservice is required to advertise its control capabilities and exposed features to the core abstraction microservice, in the form of an explicit registration providing an OpenAPI [19] specification of the offered runtime control primitives. A dedicated P&P Parser is in charge of validating the capabilities advertised by P&P plugins and control functions, basically parsing the OpenAPIs advertised (both v2.0 and v3.0 supported), and maintaining the offered primitives in the P&P Store.

The P&P Store is the persistency component and indeed it is responsible for maintaining the whole set of information required by the P&P Engine to coordinate the slice control exposure at runtime, including:

I. the customized slice exposure configuration (that is the specific instantiation of the internal slice control exposure model as provided by the P&P Manager),
II. the set of plugins and control functions registered and available,
III. the mapping (and abstraction) from primitives and capabilities offered by plugins and control functions (and included in the related OpenAPIs advertised) to the control operations and endpoints exposed to the vertical (through the vertical-tailored REST API microservice)

As said, the P&P Engine accesses these information in the P&P Store to dynamically decouple the full detailed internal slice control exposure model from the one offered to the verticals, that includes the topological view of the given slice instance augmented with runtime control operations (and related endpoints) that can be invoked (or consumed, e.g. in case of monitoring) on top of it.

At runtime, the P&P Engine consumes the slice control operations issued from the vertical-tailored REST APIs and performs all the required validations and checks (e.g. actual availability of the requested operation for the given slice) before mapping them to the proper plugin or control function according to the information kept in the P&P Store. When mapped, the REST Client in Figure 15 is used to enforce the control operation through the proper plugin or control function.

4.1.1.3 Vertical-tailored REST API microservice

The vertical-tailored REST API microservice is responsible for translating the runtime control operations exposed by the core abstraction microservice into REST operations offered to third parties (i.e. verticals), still mediated by the One-Stop-API framework, at least to augment the exposed APIs with authentication and role based access. Therefore, the vertical-tailored REST API and core abstraction microservices are strictly coupled and together implement the actual slice control exposure towards verticals and slice consumers in general.

In practice, this microservice acts as the front-end of each P&P control instance towards the One-Stop-API framework and thus the verticals space. It is basically built by two main components, as shown in Figure 16: i) a REST Controller, ii) a Continuous Monitoring Manager.

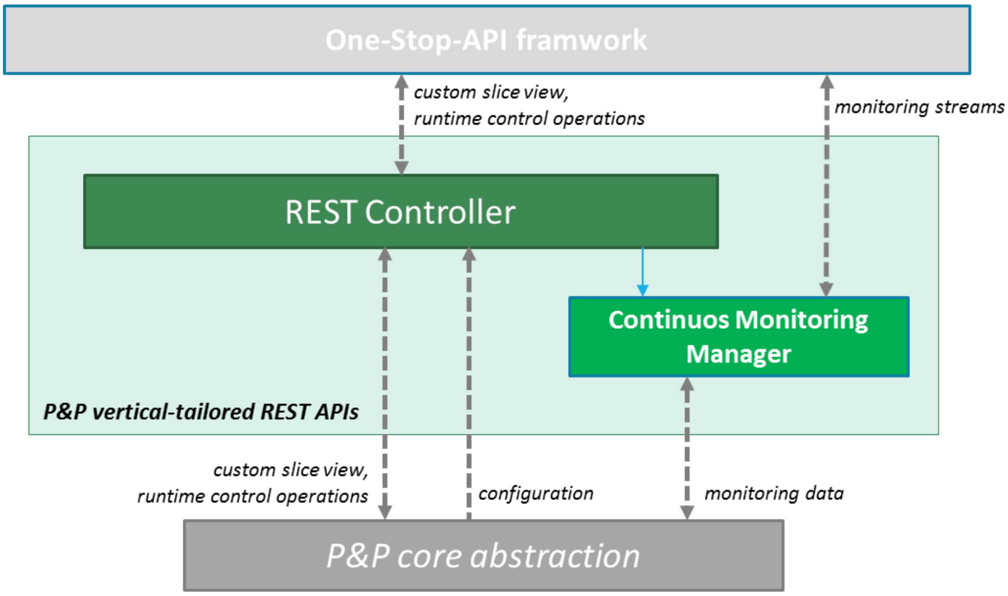


Figure 16 P&P vertical-tailored APIs microservice functional split

While the REST Controller is responsible to offer the actual REST endpoints and services as entry points for the P&P runtime slice control exposure, the Continuous Monitoring Manager allows exposing slice-specific monitoring data offered by the P&P core abstraction in the form of WebSocket streams , following the approach defined in section 4.3.2.2.

The vertical-tailored APIs offered to the One-Stop-API are mainly grouped in two main categories, as presented in section 4.3.2: i) a set of endpoints to retrieve information about the current slice control view offered to the One-Stop-API, thus including the information related to the slice elements arranged in a topology, ii) a variable set of endpoints that are the actual control operations exposed by the P&P control instance on top of the different slice elements in the topological view.

4.1.2 Plug & Play control exposure options

As said, the main key innovation brought by the P&P control framework is the exposure of dedicated control environments that enable verticals to play at runtime with their slices by means of tailored control operations and APIs while offering a customized slice view in line with vertical's requirements and business logics. The highly versatile P&P control framework approach described above allows to have heterogeneous control capabilities exposed to verticals, implemented through different combinations of core abstraction and control functions (and plugins) microservices supporting a specific specialization of the generalized slice exposure information model into the vertical-tailored slice view.

As a reference, a list of incremental control exposure levels was proposed in D2.3 [2], spanning from basic monitoring options to lifecycle management of slices and their components (e.g. network functions, virtual appliances, VNFs, MEC applications, etc.). For each level, a description of the main offered control capabilities was provided. However, due to the extensible and flexible approach followed in P&P and enabled by the microservices design choice and the pluggable control functions, the definition of a comprehensive list of control capabilities and exposure levels would somehow restrict and limit the possibilities of customization and . Indeed, the P&P control is extensible by design and the framework allows additional P&P plugins and control functions to be developed, onboarded and deployed on demand, as described in the P&P workflows in section 4.2.

In this direction, Table 1 provides a list of control capabilities that the P&P control framework is able to expose to meet the vertical requirements and business logics. For each exposed control capability, the required P&P control functions or plugins to be deployed in the context of the given P&P control instance are identified. It is implicit that these control functions and plugins need anyway to leverage on the specialization of the slice exposure view through the core abstraction microservice, as detailed in the previous sections. In practice, any combination of the control capabilities listed in Table 1 is allowed by the P&P control framework, and the selection of P&P control functions and plugins to be deployed in each P&P control instances is intended to follow the workflows defined in section and be coordinated by P&P Manager [23]. Moreover, these selection procedures will be anyway highly linked and bound to the slice orchestration principles and workflows [24], and in particular the exposed control capabilities listed in Table 1 will need to be modelled as P&P capabilities within Service Templates and Service Descriptors [16] following the approach of the Slice Template information model defined in D2.4.

An initial set of P&P control functions and plugins have been developed to fulfil the requirements of the SmartGrid use case Phase 1 [20], as detailed in sections 5.3 and 5.5. Additional exposed control capabilities, and related P&P control functions and plugins, can be however identified to fulfil more specific vertical requirements in terms of required runtime control. Moreover, part of the P&P plugins and control functions identified in the table are also envisaged to be required in support of the various single and multi-domain deployment models described in section 4.5, where interactions among different P&P control instances are enabled. This is in line with the P&P extensible approach aiming to address various and heterogeneous vertical needs.

Table 1: SliceNet P&P levels of slice control exposure

Exposed Control Capability	Required P&P control function / plugin
Monitoring of slice related metrics and KPIs, including several possible options: • Slice-level metrics • Resource level metrics • UE level metrics	• P&P plugin that hooks monitoring data as exposed by the SliceNet system (i.e. in its monitoring sub-layer [16]), and directly offer retrieved metrics to the vertical • P&P control function elaborating monitoring data exposed by the SliceNet monitoring sub-layer to produce new vertical-tailored slice metrics and KPIs, as defined in section 5.3.3

Slice QoS control, including several possible options: • selection of catalogue based QoS profiles • upgrade/downgrade of QoS attributes at network segment level (RAN, Edge, Transport, Core) • upgrade/downgrade of QoS attributes at slice level	P&P control function able to access technology agnostic SDN APIs offered by the SliceNet control plane (e.g. QoS control, forwarding graph control) and abstract them for slice level QoS oriented operations
Explicit QoE feedback enforcement to report on slice behaviour and vertical perceived performances	• P&P plugin able to interact with QoE optimization functions in the cognitive management layer • Alternatively, a P&P control function providing itself QoE optimization features and interacting with SliceNet control plane technology agnostic APIs [2] offering QoS control primitives, as defined in section 5.3.1
Control of slice User Equipments (UE) • Retrieve status of UE (registration in the mobile network, location, IP address, etc.) • Attach or detach UE to slice	P&P control function able to access UE control and management primitives exposed through the SliceNet control plane technology agnostic APIs [2] and slice management APIs (to be defined in WP6), and abstract them to offer verticals with a simple-to-use set of runtime UE operations, as defined in section 5.3.2
Network Function configuration and reconfiguration, including • catalogue based with limited configuration options • unlimited configuration options	P&P Plugin able to interact with the technology agnostic SliceNet control plane APIs and related services responsible for Network Function configuration, as defined in D2.3 [2]
Onboard and instantiation of proprietary Network Functions and virtual appliances, including: • vertical's VNFs, MEC applications • vertical specific sensors	P&P Plugin able to abstract the onboarding APIs of the Slice Service Orchestrator, as defined in D2.4 [16]
Individual Network Function lifecycle management • scale, start, stop, migration • applicable to VNFs, MEC applications, vertical's virtual appliances	P&P plugin for the abstraction of NFV/MEC/RAN Orchestrator APIs related to runtime lifecycle of Network Functions, as defined in D2.4 [16]

Slice lifecycle management • upgrade/downgrade of slice profile • full access to slice management and orchestration management	P&P Plugin for the abstraction of slice-level lifecycle management APIs offered by the Slice Service Orchestrator, as defined in D2.4 [16]

4.2 Workflows and mechanisms

This section provides a description of the workflows supported by the P&P control framework and covering the lifecycle of each P&P control instance. The aim is to describe how P&P instances are activated, updated and terminated, in terms of main actions, microservices involved and interactions with external components.

As detailed in section 4.3, the slice runtime control offered to verticals is not based on pre-determined set of control primitives and logics, but it is rather based on a customized exposure (and view) resulting from the combination (and abstraction) of the control capabilities provided by the selected P&P plugins and control functions. This means that the P&P runtime control mechanisms (in terms of workflows regulating how P&P microservices interact each other and with other SliceNet platform components) can be different from P&P instance to P&P instance, and they highly depend on the control capabilities offered (i.e. monitoring, slice elements and components runtime control, slice management). However, the high-level workflows regulating the generic slice control operations (covering also management and orchestration operations) as described in deliverable D2.3 [2]are considered as still valid and augmented with the continuous monitoring mechanism described in section 4.3.2.2.

It is important to highlight that the procedures described in the following sub-sections are intended to be coordinated by the P&P Manager component [23], which is the component in the SliceNet management layer providing the overall P&P orchestration features, as presented in deliverable D2.4 [16].

In particular, the P&P Manager is the management counterpart of the P&P control plane instances, and together they build the whole P&P framework in the SliceNet platform. The P&P Manager is intended to provide key P&P features, including the customization of the slice control exposure model into vertical-tailored slice view instances to be pushed as configurations to the P&P core abstraction microservice [16], as described in section 4.1.1.2. Moreover, following the initial high-level design of the P&P Manager included in D2.4, it is here assumed that it embeds a catalogue of the available plugins and control functions to be integrated and configured as part of P&P control instances.

4.2.1 Plug & Play control instance activation

The P&P control instance activation workflow is shown in Figure 17, and it describes the procedure for creation and configuration of a new P&P control instance to be offered to a given vertical and providing a dedicated and tailored slice control environment.

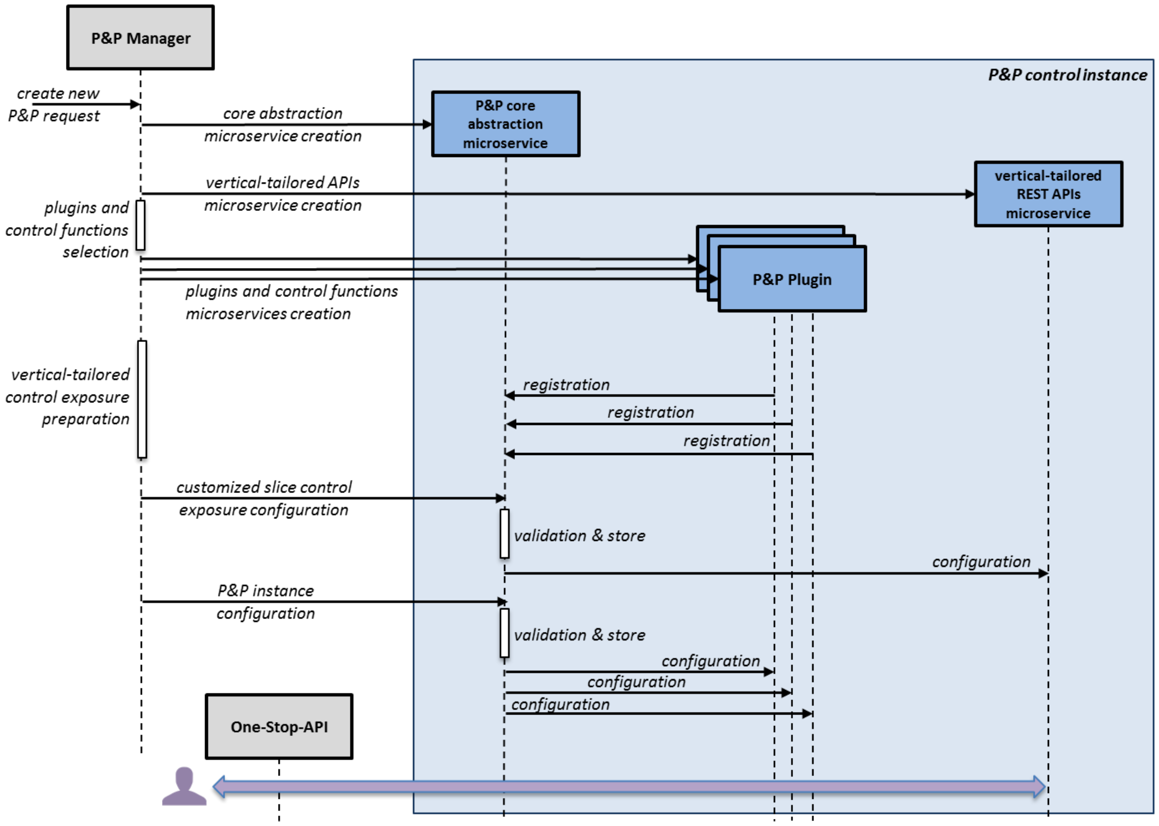

P&P Manager
create new P&P request
core abstraction microservice creation
P&P control instance
P&P core abstraction microservice
vertical-tailored APIs microservice creation
vertical-tailored REST APIs microservice
plugins and control functions selection
P&P Plugin
plugins and control functions microservices creation
vertical-tailored control exposure preparation
registration
registration
registration
customized slice control exposure configuration
validation & store
configuration
P&P instance configuration
validation & store
configuration
configuration
configuration
One-Stop-API

Figure 17 P&P control instance activation workflow

The workflow is mostly driven and coordinated by the P&P Manager, and it is implemented through the following main steps:

- The P&P Manager is requested to deploy a new dedicated P&P control instance for a new slice provisioned and offered to a vertical. Following the P&P Manager's high-level design in D2.4 [16], this operation is requested by the Service and Slice Orchestrator.
- The P&P Manager creates a new instance of the P&P core abstraction and vertical-tailored REST APIs microservices. These can be launched as containerized applications, as described in section 5.4
- The P&P Manager selects the proper set of plugins and control functions to be added to the P&P control instance in order to fulfil the slice control exposure agreed and requested by the vertical
- The P&P Manager creates a new instance of each of the selected P&P plugins and control functions, as containerized applications in line with what described in section 5.4
- Once the P&P plugins and control functions microservices are up and running, each of them advertise its control capabilities and APIs to the core abstraction microservice. This is done by providing an OpenAPI specification, as described in section 4.3.2.1
- The P&P Manager prepares the vertical-tailored slice view, again according to the control exposure agreed with the vertical, following the internal slice exposure information model described in section 4.3.1.1. This tailored slice control view, including the slice topology and the set of control primitives exposed, is then provided to the P&P core abstraction microservice as configuration through the interface specified in section 4.3.2
- The P&P core abstraction microservice validates the received customized slice view configuration, basically cross-checking what is offered by the plugins and control functions available and thus storing the mapping with the control operations exposed to verticals. In addition, the vertical-tailored REST APIs microservices is configured with the list of control endpoints to be exposed through the One-Stop-API framework

- Finally, the P&P Manager issues the specific P&P control instance configuration to the P&P core abstraction microservice, through the interface described in section 4.3.2.1. This configuration is validated and processed and then each plugin and control function is in turn configured as described in section 4.3.3
- At this point, the P&P control instance is up and running and ready to be used by the given vertical through the One-Stop-API

4.2.2 Plug & Play control instance runtime update

Each P&P control instance can be subject to runtime updates for mainly two reasons: i) the related slice instance has been updated (e.g. in its logical topology) and the exposed slice view is affected, ii) a new plugin or control function has to be deployed as part of the P&P control instance in support of an updated slice control exposure to be offered to the vertical. It is important to highlight that in both cases the microservices approach used (with orchestrated containerized applications as described in section 5.4) allows that the instance is updated at runtime without any P&P service disruption perceived by the vertical.

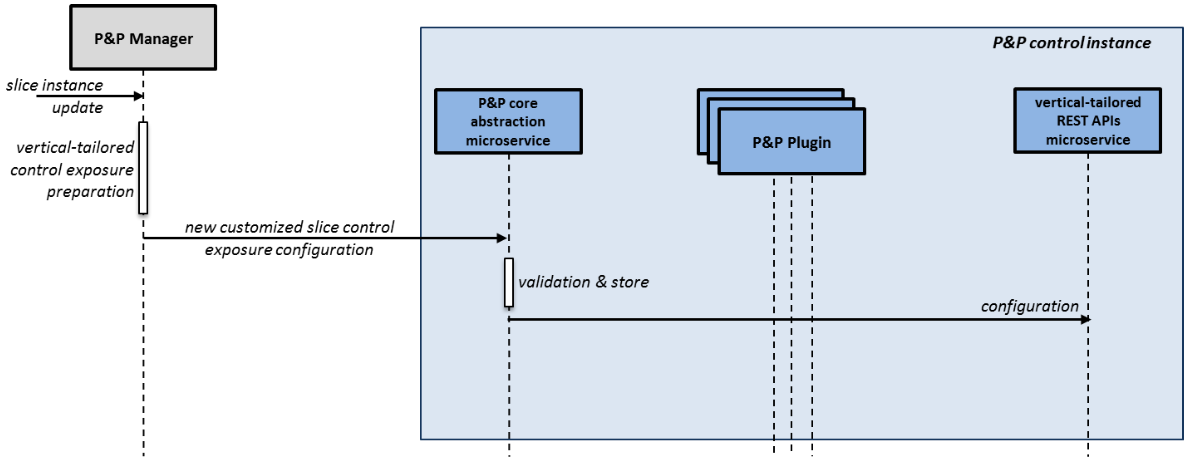

P&P Manager
slice instance update
vertical-tailored control exposure preparation
new customized slice control exposure configuration
P&P control instance
P&P core abstraction microservice
P&P Plugin
vertical-tailored REST APIs microservice
validation & store
configuration

Figure 18 P&P control instance runtime update workflow – new vertical-tailored slice view

The workflow related to the runtime update of the vertical-tailored slice view is implemented through the following main steps:

- The P&P Manager is informed that the slice instance has been updated in one or more of its constituent elements (e.g. topology, location of network functions, etc.). This can be done either through a notification or an explicit trigger and will be detailed as part of the P&P Manager specification [23]
- The P&P Manager checks if the slice control exposure agreed with the vertical is affected by the slice instance update, and in case it prepares an updated vertical-tailored slice view, including the new slice topology and, if required, the updated set of control primitives exposed
- The P&P Manager issues the new customized slice control exposure configuration to the P&P core abstraction microservice, through the interface described in section 4.3.2
- The P&P core microservice validates the new customized slice view against the information model reported in section 4.3.1.1, and then updates the configuration of the vertical-tailored REST APIs microservice to expose (if required) any new control primitive to the vertical
- At this point, the P&P control instance is updated and the exposed slice view is aligned with the current status of the slice instance

On the other hand, in some cases there might be the need of updating the P&P control instance by deploying new P&P control functions in support of new control capabilities to be exposed to the vertical. This may happen when the vertical agrees with the slice provider to enhance (or generally update) its slice control exposure with new control capabilities to be offered through an additional P&P control function. This case, with respect to the previous runtime update, is more related to off-line interactions between the vertical and the slice provider as part of dynamic slice capabilities negotiation. In this case, the workflow is implemented through the following steps:

- The P&P Manager is requested to update at runtime a P&P control instance. This can be done through explicit trigger and will be detailed as part of the P&P Manager specification [23]
- The P&P Manager selects the proper set of new plugins and control functions to be added to the P&P control instance in order to fulfil the updated slice control exposure
- The P&P Manager creates a new instance of each of the selected P&P plugins and control functions, as containerized applications in line with what described in section 5.4
- When the new P&P control function microservice is up and running, it advertises its control capabilities and APIs to the core abstraction microservice
- At this point, the P&P Manager prepares the new vertical-tailored slice view, taking into consideration the new slice control exposure to be offered, following the internal slice view information model described in section 4.3.1.1.
- The new tailored slice control view is issued to the P&P core abstraction microservice as configuration through the interface specified in section 4.3.2
- The P&P core abstraction microservice validates the new slice view configuration, and it updates the mapping between control primitives exposed to the vertical and control capabilities from the plugins according to the new features offered by the new control function.
- The new P&P control function and the vertical-tailored REST APIs microservice are then configured following the same principles of the P&P control instance activation workflow
- At this point, the P&P control instance is updated the slice control exposure is aligned with the new P&P capabilities

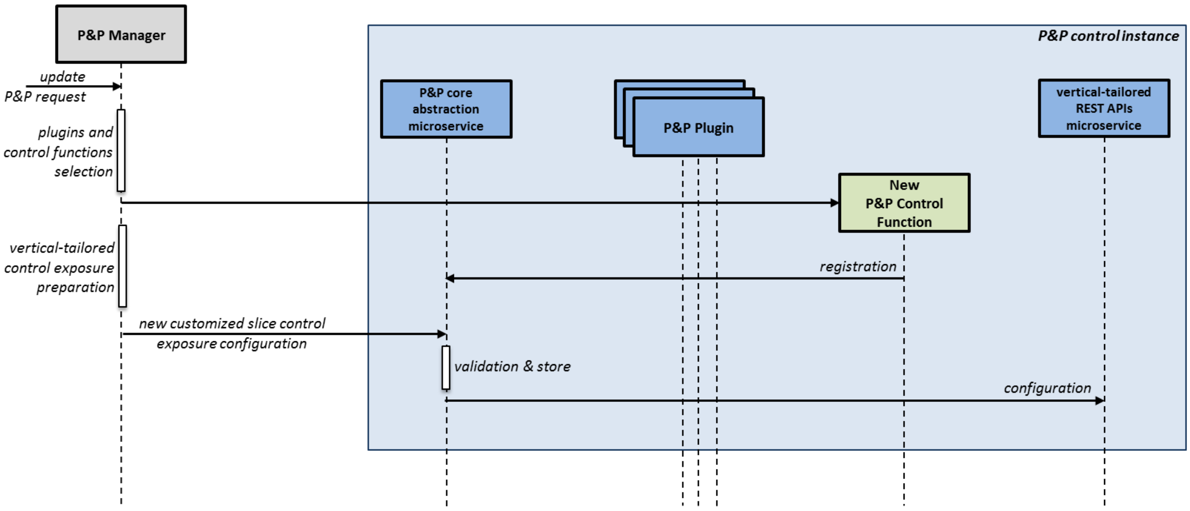


Figure 19 P&P control instance runtime update workflow – new plugin or control function added

4.2.3 Plug & Play control instance termination

The P&P control instance termination workflow is presented in Figure 17 in the form of a sequence diagram, and it presents how a running P&P control instance and the related microservices are terminated. The workflow is again driven by the P&P Manager, and it is implemented through the following main steps:

- The P&P Manager is informed that the slice instance is going to be terminated. This can be done either through a notification or an explicit trigger and will be detailed as part of the P&P Manager specification [23]
- The P&P Manager issues a deletion of the slice control exposure configuration to the P&P core abstraction microservice. This is interpreted as a request to deactivate all the control exposure features and, after a check that there are no pending control operations requested by the vertical, a deletion of the current configuration of the vertical-tailored REST APIs microservice is issued to stop exposing runtime control primitives to the vertical
- At this point, the P&P Manager coordinates the termination of all the microservices composing the P&P control instance, thus by stopping all the related applications
- The P&P control instance is terminated and all the related slice control exposure features are no more available

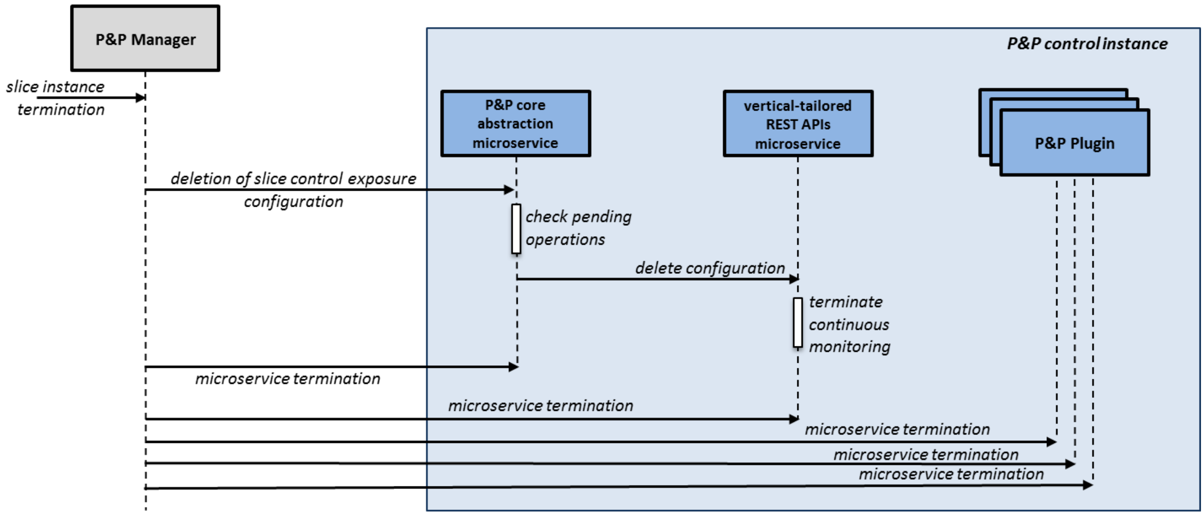


Figure 20 P&P control instance termination

4.3 Information models and interfaces specifications

4.3.1 Slice exposure information model and principles

P&P is an innovative framework introduced by SliceNet to enable flexibility with respect to the accommodation of different vertical needs without restricting customer views to a limited set of predefined layouts and API endpoints.

As a consequence, the slice exposure information model pertaining to P&P has evolved along the same principles. More specifically, aiming that slicing should be tailored to the vertical needs, the slice exposure information model has to be generic enough to cater for any topological definition of the slice view. The customized view is produced starting from the control exposure requirements of the vertical as expressed in the slice template used as reference for the provisioning of the given slice instance, as described in deliverable D2.4 [16]. For instance, exposure in certain vertical cases should be enough to contain only the slice endpoints (User Equipment, backend servers and application specific entities as databases) and the logical linking among them. On the contrary, logical endpoints and logical connections might not be enough for Service Provider oriented slice consumers (e.g. a DSP) where a more detailed decomposition containing slice communication, functional and resource elements may be required. This also maps to the considerations reported in section 4.5 for what concern the suitability and validity of the P&P control framework for any slice consumer to slice provider interaction, being it from NSP to DSP, from DSP to vertical or from NSP directly to vertical.

This generalization of the slice exposure model implies consequently that it should be at the same time an abstraction of the overall SliceNet information model (as reported in D2.4 [16]), but agnostic to the slicing technology and able to address the description of any level of the SliceNet information model from vertical service, to slice, and down to network functions and resources levels. Therefore, at any level, the slice topology view and elements have to be aligned to the level of the intended exposure, with the vertical customization being adequate to describe the type of the constituent elements, e.g. RAN or CORE functions, User Equipment, application servers, VNFs, MEC functions.

Apart from the slice view that the P&P is producing, a set of runtime control operations is foreseen to be offering the customized slice control, management and monitoring options that the vertical should have at its disposal according to the slice template [16]. The plugins and control functions deployed in each P&P control instance enable a vertical related control, management, and monitoring set of operations to be specially created and exposed as dedicated control endpoints.

As anticipated in section 4.1.1.2, the P&P slice exposure information model itself includes two different sub-models, covering the internal slice control view and the northbound slice control view. The former aims to be utilised internally by the P&P control instance and it drives the internal P&P logic implemented by the core abstraction microservice. It allows the building of the customized control view of the slice for the consumer on the one hand, while at the same time allows for binding to specific control actions offered by the P&P plugins and control functions. The northbound slice control view model is an augmented version of the internal model, and provides the actual slice control view exposed to the slice consumer through the P&P northbound APIs, including the definition of the actual runtime control endpoints accessible by the vertical.

As described in section 4.2, the customization of the slice exposure model according to the vertical requirements is performed by the P&P Manager entity that is invoked during the slice provisioning phase [16].

4.3.1.1 Plug & Play internal slice exposure information model

The P&P internal slice exposure information model is shown in Figure 21. As said, it is basically composed by a topological view of the slice in the form of a graph, where generic slice elements are interconnected by means of logical links. The specialization of each slice element with a vertical-tailored slice context is the core part of the slice exposure customization. Indeed, for each element in the view, a list of attributes allows to define its specific slice-context, practically identifying the level of granularity of the overall slice view and elements themselves, spanning from resource (e.g. VNFs, MEC applications, RAN) to slice (e.g. User Equipment, 4G/5G network functions) and service (e.g. vertical's specific appliances) levels. Moreover, for each element, an additional set of properties allows to identify the list of control APIs offered by P&P plugins and control functions on top of the element itself, and that can be leveraged by the vertical during its runtime control operations.

A formal description of the internal slice exposure information model is provided in Table 2 and subsequent tables, which detail the structure of each component and sub-components of the model. Moreover, the JSON schema related to this internal slice exposure model is provided in Annex A.1, together with an example of instantiation in Annex A.2.

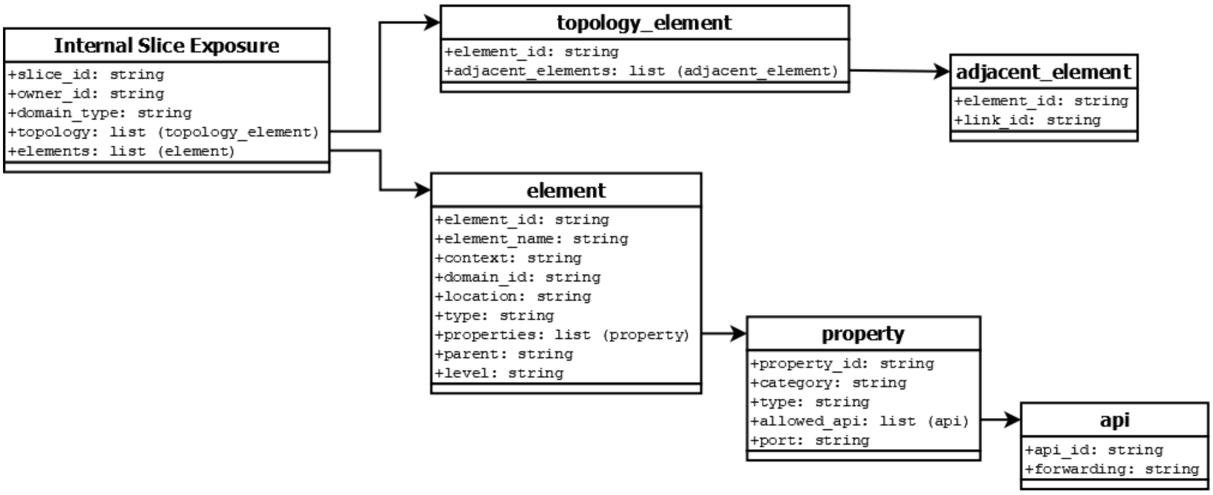


Figure 21 Plug & Play internal slice exposure information model

Table 2 P&P internal slice exposure model

Identifier	Cardinality	Description
slice_id	1	It is the unique ID of the given slice instance
owner_id	1	It uniquely identifies the owner of the slice, from the consumer perspective.
domain_type	1	It indicates whether the given slice is single or multi-domain. It can therefore be "single" or "multi"
topology	1	It provides the topological view of the slice in the form of a connected graph. It follows the structure in Table 3
elements	1	It is the list of slice elements building the customized slice view. Each element can be either a node or a link in the topology above. The structure of each element follows Table 5

Table 3 P&P internal slice graph model

Identifier	Cardinality	Description
element_id	1	It is the ID of a given slice element in the slice view topology. This element_id refer to an element of Table 5, in particular of type "node".
adjacent_elements	0…1	It is the list of adjacent elements in the slice view, and it follows the structure in Table 4

Table 4 P&P internal slice adjacency model

Identifier	Cardinality	Description
element_id	1	The ID of the adjacent element in the slice view topology, connected to another element as per Table 3
link_id	1	The ID of the logical link connecting the two adjacent elements in the slice view

Table 5 P&P internal slice element model

Identifier	Cardinality	Description
element_id	1	The unique ID of the slice element
context	1	It identifies the slice context of the element in the slice view. It can be: “UE, “RAN”, “CORE”, “VNF”, “PNF”, “NS”, “MECApp”, “APP”, “VL”, “INTERDOMAIN”, “TRANSPORT”
element_name	1	A user friendly name of the element in the slice view.
domain_id	1	It identifies the domain the slice element belongs to.
location	1	It identifies the location where the slice element resides or is deployed (e.g. “edge-xyz”)
type	1	It is the type of slice element. It can be “link” or “node”
properties	0..1	It is the list of additional properties of the slice element, used to described the control APIs that can be invoked on this element. See Table 6
parent	0..1	It is the identifier of the parent slice element, to be used when the slice view is hierarchical.
level	0..1	The level of granularity and abstraction of this slice element. It can be: “Service”, “Slice”, “Subslice”, “NF”.

Table 6 P&P internal slice element properties model

Identifier	Cardinality	Description
property_id	1	It identifies a given slice element property, and it maps to a plugin or control function in the P&P

		control instance.
category	1	It identifies the category of the additional property of the given slice element. It can be: “management”, “control”, “monitoring”
port	0..1	It is the port to be used to access the control APIs defined in the “allowed_api” field.
allowed_api	0..1	It identifies the list of control APIs exposed by the given plugin or control function and associated to this slice element. See Table 7

Table 7 P&P NBI Element API Model

Identifier	Cardinality	Description
api_id	1	It is the reference to the identifier of the given API operation in the plugin or control function OpenAPI specification (that has been provided to the P&P core abstraction during its registration)
forwarding	1	It is the mapping applied to this API operation towards the vertical. If “direct”, the vertical can invoke the API as it is provided by the plugin or control function

4.3.1.2 Plug & Play northbound slice exposure information model

The information model for the northbound slice exposure is aimed to be used for the provision of the ergonomic features of an adaptable control interface at the disposal of the vertical or slice consumer. In particular, for each P&P control instance, an instantiation of the northbound slice exposure model is automatically generated by the P&P core abstraction starting from the internal slice view. This northbound slice view is then exposed to the vertical through the One-Stop-API vertical by means of a dedicated API as described in section 4.3.2.1.

The structure of this model (shown in Figure 22) is similar to the one of the internal slice exposure, while it provides additional details suitable for direct exposure to verticals and hides the capabilities of P&P plugins and control functions. Indeed, the key point of differentiation is that this slice exposure model also includes the list of actual runtime control endpoints and APIs that the vertical can invoke on each of the slice element. These runtime control endpoints are not pre-defined and are dynamically created by the P&P core abstraction microservice according to the customized slice exposure the vertical has agreed with the slice provider. Moreover, following an OpenAPI-like approach, explicit definitions of requests and responses body schemas are included as part of the northbound slice view.

As described in section 4.3.2.3 this approach for exposing runtime control APIs to verticals as part of the northbound slice view is aligned with emerging REST application architectures where servers provide information dynamically to clients through hypermedia, thus avoiding REST clients to have full knowledge about how to interact with servers (e.g. in terms of pre-defined endpoints).

A formal description of the northbound slice exposure information model is provided in Table 8 and subsequent tables, which detail the structure of each component and sub-components of the model.

Moreover, the JSON schema related to this internal slice exposure model is provided in Annex B.1, together with an example of instantiation in Annex B.2.

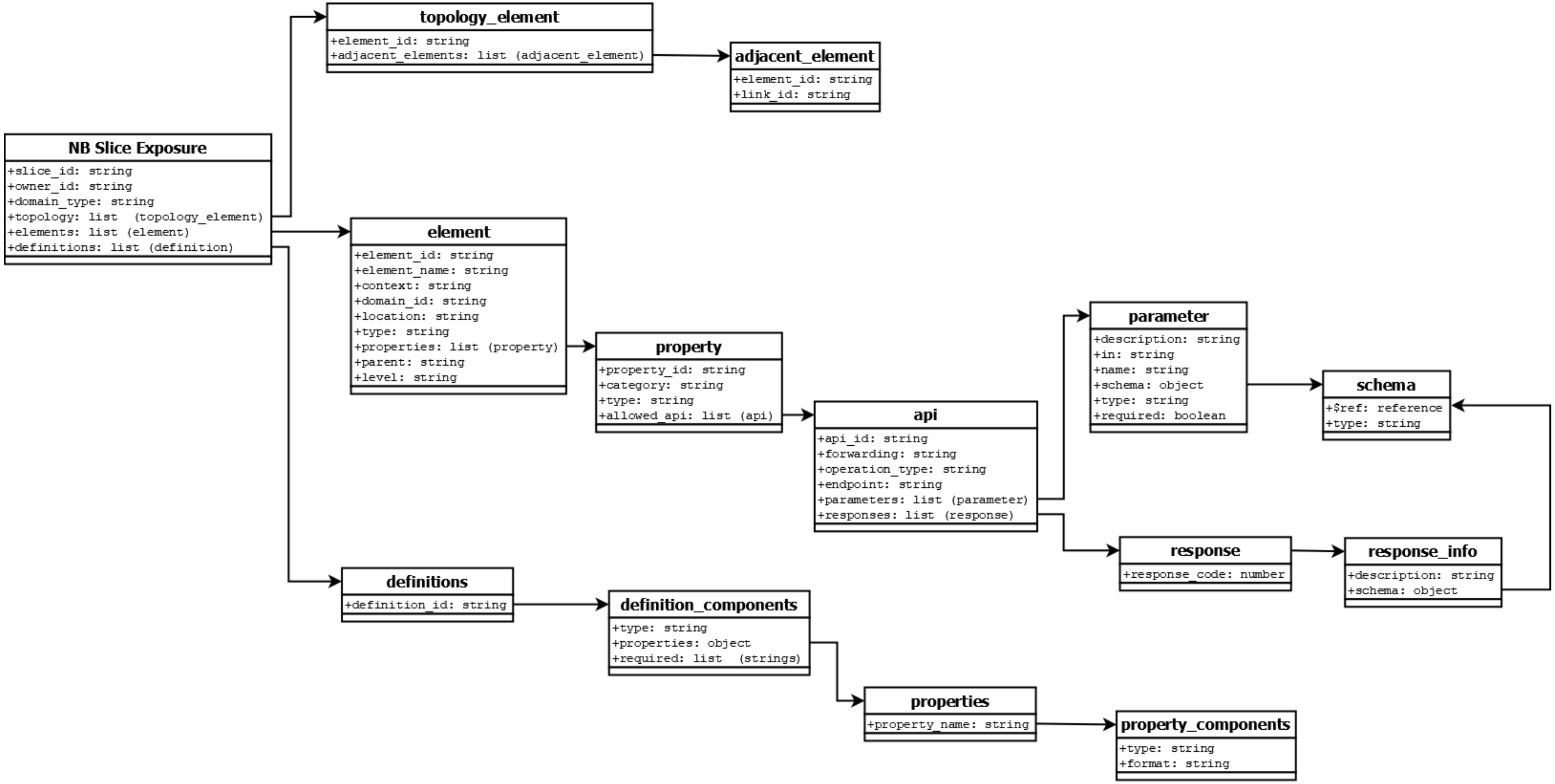


Figure 22 Plug & Play slice external model

Table 8 P&P northbound slice control view

Identifier	Cardinality	Description
slice_id	1	It is the unique ID of the given slice instance. It is the same as
owner_id	1	It uniquely identifies the owner of the slice, from the consumer perspective.
domain_type	1	It indicates whether the given slice is single or multi-domain. It can therefore be “single” or “multi”
topology	1	It provides the topological view of the slice in the form of a connected graph. It follows the same structure of the internal slice exposure in Table 4.
elements	1	It is the list of slice elements building the customized slice view. Each element can be either a node or a link in the topology above. The structure of each element follows Table 9.
definitions	0..1	A list of JSON objects referred to as schema definitions of the payloads exchanged (for both requests or responses) via the northbound P&P REST APIs. The item of this list are produced from the OpenAPI specification of plugins and control functions. See Table 15.

Table 9 P&P NBI Element Model

Identifier	Cardinality	Description
element_id	1	The unique ID of the slice element
context	1	It identifies the slice context of the element in the slice view. It can be: “UE, “RAN”, “CORE”, “VNF”, “PNF”, “NS”, “MECApp”, “APP”, “VL”, “INTERDOMAIN”, “TRANSPORT”
element_name	1	A user friendly name of the element in the slice view.
domain_id	1	It identifies the domain the slice element belongs to.
location	1	It identifies the location where the slice element resides or is deployed (e.g. “edge-xyz”)
type	1	It is the type of slice element. It can be “link” or “node”

properties	0..1	It is the list of additional properties of the slice element, used to described the control APIs that can be invoked on this element. See Table 10
parent	0..1	It is the identifier of the parent slice element, to be used when the slice view is hierarchical.
level	0..1	The level of granularity and abstraction of this slice element. It can be: “Service”, “Slice”, “Subslice”, “NF”.

Any element in the northbound slice view information set can have a set of properties that identify the control exposure. These details are important for automatically creating the interaction features through which the user can interact with the slice.

Table 10 P&P NBI Element Properties

Identifier	Cardinality	Description
category	1	It identifies the category of the additional property of the given slice element. It can be: “management”, “control”, “monitoring”
allowed_api	1	It identifies the list of control APIs exposed towards the vertical and associated to this slice element. See Table 13

The API model listed next describes the REST based web service details through which the vertical interacts with the slice.

Table 11 P&P NBI Element API Model

Identifier	Cardinality	Description
api_id	1	It is the reference to the identifier of the given API operation in the plugin or control function OpenAPI specification (that has been provided to the P&P core abstraction during its registration)
operation_type	1	The type of operation that has to be used for invoking the service, choice among: “post”, “get”, “put”, “patch”, “delete”
endpoint	1	The URI related to the given control operation exposed at northbound for the given slice view element
parameters	0..1	A list of parameters relating to the service invocation, see Table 12
responses	1	A list of object representing the possible query replies. See Table 13Table 13 P&P NBI API Response code

While the API Model above describes the way to interact with the P&P, the following parameter model identifies the content that has to be used. This part of the model is required to build the user dialog interfaces.

Table 12 P&P NBI Element API Parameters Model

Identifier	Cardinality	Description
description	1	A description of the parameter
in	1	The position of the parameter value. It can be either “body”, or “path” if it has to be part of the request URI
name	1	The name of the parameter to be used in the request, either as key property in the body or as part of the URL query name
schema	0..1	In case of a complex structure, an OpenAPI definition of it is appended under a specific object reference in the definitions section. See Table 15
required	1	A boolean flag (true/false) indicating if the parameter is mandatory for the service invocation or optional

As the replies to the requests might be also complex, a similar approach based on OpenAPI definitions is followed also for complex replies. The details are elaborated in the following tables.

Table 13 P&P NBI API Response code

Identifier	Cardinality	Description
response_code	1	The HTTP response code of query reply, following the structure in Table 14

Table 14 NBI API Response code information

Identifier	Cardinality	Description
description	1	Description of HTTP response code (e.g. “OK”, “Not Found”, etc.)
schema	0..1	It contains the reference to the reply payload of the query, if any. The structure of schema is the same of the one used in “parameters” (Table 11) and defined in Table 15

Table 15 P&P NBI Element API Service Parameters Schema model

Identifier	Cardinality	Description
$ref	1	Reference to the schema (e.g. “#/definitions/{definition_id}”). $ref points to an item stored in “definitions” list. See Table 8

type	0…1	Type of reference. For instance, if type = “array”, the schema will be an array of object referenced by $ref. If present, schema structure becomes as follows:
		`"schema": {`
		`   "type": "array",`
		`   "items": {`
		`      "$ref": "#/definitions/{definition_id}"`
		`   }`
		`}`

Table 16 P&P NBI Definition identifier

Identifier	Cardinality	Description
definition_id	1	It is the identifier of the schema that is a complex object described in Table 17.

Table 17 P&P NBI Definition model

Identifier	Cardinality	Description
type	1	It is the type of the schema. The default value is “object”.
properties	1…N	A list that specifies each field of the schema in term of name, type and format. See Table 18.
required	0..1	A list of property names that have to be specified in query payload.

Table 18 P&P Component model property identifier

Identifier	Cardinality	Description
property_name	1	Name of a field in the schema. The field is described in Table 19

Table 19 P&P Component model property schema

Identifier	Cardinality	Description
type	1	It is the type of the field. JSON schema defined types are: “string”, Numeric Types (e.g. “integer”), “object”, “array”, “boolean” and “null”

format	0..1	Fixes, if needed, the format of the type. An example:
		`{`
		`    “type”: “integer”,`
		`    “format”: “int64”`
		`}`

4.3.2 Plug & Play northbound APIs

The P&P northbound APIs are defined as the whole set of REST APIs that a given P&P control instance exposes, either for management (internal or external) purpose or for enabling a tailored runtime control to verticals. These REST APIs can be grouped into three main categories:

- Management APIs, representing the fixed set of endpoints and operations exposed by each P&P control instance to configure, update and retrieve the tailored slice view following the models defined in section 4.3.1
- Continuous monitoring APIs, to expose towards verticals (through the One-Stop-API) continuous data streams including tailored slice performance metrics and statistics
- Runtime control APIs, that is the collection of tailored control APIs exposed towards verticals and dynamically created according to the customized slice view and the P&P capabilities in terms of control functions and plugins

The runtime control APIs map to the high level P&P interfaces defined in D2.3 [2], and are those intended to provide the second level of abstraction in the SliceNet control plane paradigm, thus exposing towards verticals a customized and dedicated per-slice view with runtime control endpoints.

It is worth to highlight that authentication and role based access functionalities on top of the P&P northbound APIs, especially for those related to runtime control and continuous monitoring, are intended to be provided by the One-Stop-API framework, as the layer in the SliceNet architecture regulating the interactions among different business roles.

4.3.2.1 Management APIs

The P&P northbound REST interface offers a set of common management operations, as they are available for every P&P instance at the same URL paths. Such operations enable the P&P Manager to configure the vertical tailored slice view on the one hand (following the workflows defined in section 4.2) and the verticals (through the One-Stop-API) to retrieve the current northbound slice view and the available runtime control endpoints operations and API. The following tables provide the details for these operations.

Table 20 P&P Slice view configuration API details

Endpoint	`/plug-and-play/pp_management/registration/slice/`

HTTP Verb	POST
Description	Configuration of the vertical-tailored (internal) slice view on the P&P core abstraction microservice. This operation is also used for dynamic P&P control instance updates following the workflow described in section 4.2.2.
Caller	P&P Manager
Request Body	Internal slice view following the model in section 4.3.1.1 and the schema in Appendix A.1. Format: JSON
Response Body	None
Response Codes	200 OK - The slice view has been successfully configured in the P&P

Table 21 P&P Instance configuration API details

Endpoint	`/plug-and-play/pp_management/registration/pp_config/`
HTTP Verb	POST
Description	Configuration of the P&P control instance, with a set of configuration that is later partially forwarded to plugins and control functions.
Caller	P&P Manager

Request Body	JSON object with the following structure:
	{
	    “<type_service_1>”: [“key1”, “value1”, …, “keyN1”, “valueN1”],
	    “<type_service_2>”: [“key2”, “value2”, …, “keyN2”, “valueN2”],
	    …
	}
	where *<type_of_service>* identifies the specific category of the configuration set, that can differ to fulfil the requirements for accessing the SliceNet platform components with P&P plugins and control functions. Thus it can be: “control”, “management”, “monitoring”.

Response Body	None
Response Codes	200 OK - The P&P configuration has been successfully stored in the P&P core

Table 22 P&P control instance slice view API details

Endpoint	`/plug-and-play/pp_management/get-info/{info_type}`

HTTP Verb	GET
Description	Retrieve information about the P&P instance tailored slice view. Possible values of {info_type}: • **modules**: detailed list of registered plugins and control functions • **slice:** current internal slice view as configured by P&P Manager • **topology:** current slice view topology graph • **detailed-slice:** current northbound slice view augmented with the runtime control endpoints available (compliant with the model defined in section 4.3.1.2), as offered to the vertical through the One-Stop-API
Caller	Vertical through the One-Stop-API P&P Manager One-Stop-API (for internal purposes)
Request Body	None
Response Body	Variable structure according to *{info_type}* and compliant with information models defined in section 4.3.1. Format: JSON
Response Codes	200 OK 404 Not Found – Information about selected element have not been found

4.3.2.2 Continuous monitoring APIs

The access to slice monitoring information can be considered as a basic control exposure feature allowing verticals to be aware of statistics and performances related to their slices. The nature of the monitoring data accessible is subject to the P&P control customization process, and it is expected to be (partially) regulated and described as part of the vertical service and slice templates and descriptors [16]. This means that all resource, network function, slice or service level monitoring information can be in principle exposed to verticals according to the agreed control exposure.

The P&P control framework described in this document allows to expose slice related monitoring information as continuous streams of data. The base idea is to transform single-query monitoring operations offered, usually a GET, in a periodic-executed query that pushes results on proper channel, creating a data stream. The technology used to create such channel is based on the concept of WebSocket [22].

Each P&P instance is able to handle one or more websockets in order to meet the requirement of having multiple streams per slice. In any case, there will be a single websocket per active data stream user. First, the data stream user (e.g. the One-Stop-API) should open the websocket session with the P&P instance, which acts as websocket server. As reply to the open request, if it successes, the data stream user receives a numeric code representing the ID of the websocket assigned to that user: this ID shall be used on every later data stream request for that websocket session.

Once the websocket is opened, a vertical can request through the One-Stop-API the continuous monitoring of desired performance metric (e.g. through a One-Stop-API graphical user interface), on condition that such metric can be retrieved by a GET runtime control operation already exposed by the given P&P instance.

In practice, a vertical can request the P&P instance to periodically execute a given monitoring query and, then collect the results through the opened websocket as continuous data streams. The vertical can perform such continuous monitoring requests through the operation described in Table 23 below.

Table 23 P&P Core continuous monitoring request API details

Endpoint	`/plug-and-play/websocket/{ws_id}/{element_id}/{api_id}/`
HTTP Verb	PUT
Description	This operation allows is used to request the P&P core for perform one shot monitoring operation periodically and put the results on specific websocket • **ws_id:** the id of the websocket • **element_id:** the id of the element in the northbound slice topological view that exposes the target monitoring operation • **api_id:** the id of the target operation in the northbound slice view
Caller	Vertical through the One-Stop-API
Request Body	None
Response Body	None
Response Codes	200 OK – Continuous monitoring successfully configured 404 Not Found – No opened websocket session has been found

Since a single websocket can be opened per continuous monitoring session, and the vertical can request multiple data streams, the P&P core abstraction microservice augments each message in the data stream with proper attributes, in order to make the consumer able to distinguish different data streams and monitoring information structures. An example is provided in Snippet 1 below.

```
{
    "slice_id": "slice-001",
    "owner_id": "tenant-001",
    "element_id": "element-001",
    "api_id": "get-slice-qos",
    "content": {
            <PAYLOAD of GET REPLY>
    }
}
```

Snippet 1 Format of single continuous monitoring message

The "content" field conveys the reply of the monitoring query (according to the schema included in the northbound slice view, see "definition" attribute in Table 8), while the "slice_id", "owner_id", "element_id" and "api_id" fields guarantee to make a data stream unique across different slices context at the One-Stop-API side.

4.3.2.3 Runtime control APIs

One of the main innovative features of P&P control framework is that the vertical-tailored control APIs exposed to verticals through the One-Stop-API layer are not bound to specific pre-defined set of endpoints or resources. As described in the previous sections, this is enabled by the way the P&P

slice exposure information models are defined and conceived. In particular, the P&P control framework follows an approach where client and server sides of the REST API are decoupled and interact by means of dynamic hypermedia information provided by the server, similarly to the REST Hypermedia As The Engine Of Application State (HATEOAS) approach [25]. In practice, the client side of the API (i.e. the vertical through the One-Stop-API) accesses the REST application (i.e. the P&P control instance) through a simple pre-defined endpoint offered by the server (i.e. the P&P management API defined in Table 22). After that, all other interactions are based on information (i.e. endpoints and resource representations) that the client discover directly from the server through dynamic hypermedia. In the case of P&P control instance, this hypermedia is represented by the set of allowed APIs properties within each element in the slice view, as defined in section 4.3.1.2. This approach enables much more flexibility and extensibility if compared with traditional service oriented architectures where REST interactions happen over pre-defined and fixed interfaces shared through documentation or other out-of-band mechanisms.

For this reason, it is not possible to derive in advance the full list of runtime control REST APIs that a given P&P control instance is able to expose. As said, this strictly depends on the customized way the slice is exposed, i.e. how the slice exposure models defined in section 4.3.1 are specialized with tailored slice-context information and how the P&P instance is composed in terms of control functions and plugins. This also gives an unprecedented flavour of extensibility to the P&P control framework, that can be very easily adapted to any vertical requirement in terms of expected runtime control granularity on the one hand, as well as in terms of slice exposure constraints set by slice providers.

As an example of this approach, Annex B.2 provides a vertical-tailored northbound slice view for a SmartGrid slice, where each slice element in the view includes dedicated hypermedia information related to REST APIs available to control at runtime the slice itself.

4.3.3 Common plugins and control functions APIs

As said, each P&P control function and plugin is a dedicated microservice aiming to implement a specific and tailored logic, either to be exposed to verticals or to be used by other control functions and plugins part of the same P&P instance. The communication between core abstraction and plugins is based on a REST interfaces each element exposes. In particular, the P&P core abstraction exposes a REST API (which is kept internal to the P&P instance and not exposed to northbound), with same endpoint in each P&P instances, used by plugins and control functions to register themselves to the P&P core abstraction.

Table 24 P&P core abstraction plugin registration API details

Endpoint	*/plug-and-play/pp_management/registration/module/*
HTTP Verb	POST
Description	It enables plugins and control functions to register to P&P core abstraction microservice
Caller	Plugin
Request Body	REST interface description provided in Swagger2.0 or OpenAPI 3.0. Format: JSON. See Annex C, D and E for examples
Response Body	None

Response Codes	200 OK – Module successfully registered on P&P core

On the other hand, each plugin has to expose a dedicated REST API callable by the core abstraction for receiving specific configuration information. Such API shall be provided by each plugin and control function as part of the OpenAPI specification advertised to the P&P core abstraction at the registration phase. Being it an internal API, it is not then remapped on P&P northbound APIs by the P&P core abstraction.

Table 25 Plugin configuration API details

Endpoint	*<Defined by plugins and control functions and included in their advertised OpenAPI>*
HTTP Verb	POST
Description	Enforcement of P&P plugin/control function configuration. It follows a similar structure to the API exposed by the P&P core abstraction for configuration purposes and described in Table 21
Caller	P&P core abstraction

Request Body	JSON object with the following structure:
	`{`
	`    “<type_service_1>”: [“key1”, “value1”, …, “keyN1”, “valueN1”],`
	`    “<type_service_2>”: [“key2”, “value2”, …, “keyN2”, “valueN2”],`
	`    …`
	`}`
	where *<type_of_service>* identifies the specific category of the configuration set, that can differ according to the category of the plugin or control function. Thus it can be: “control”, “management”, “monitoring”. For each category it is expected that a different configuration set is required to fulfil the requirements for accessing the related SliceNet platform components.

Response Body	None
Response Codes	200 OK – Plugin successfully configured

Once a plugin receives the configuration from the core abstraction, it uses the information to access the correspondent SliceNet platform component according to the nature of the runtime control operation (i.e. control, monitoring, management, orchestration). For instance, a P&P plugin or control function that has the purpose to provide control capabilities and thus the need of interacting with one or more SliceNet control plane services (e.g. for QoS control) through the technology agnostic APIs [2], will receive a configuration as detailed in the Snippet 2 below.

```
{
    “control”: {“cpsr_ip”, “x.y.z.t”, “cpsr_port”, “12345”}
}
```

Snippet 2 Example of P&P plugin or control function configuration

where, a set of information required to access the Control Plane Service Registry (CPSR) component within the SliceNet Control Plane is provided [26], being the CPSR responsible to provide control plane service discovery features according to the service based architecture defined in D2.3 [2].

4.4 Plug & Play features for QoE optimization

One of the key SliceNet innovations is to provide vertical informed quality of experience (QoE). In brief, the idea behind this concept is to allow verticals to provide some kind of feedback about the performance perceived by them as users of the system; the so-called QoE. Since the P&P is the entity responsible for providing the vertical with control capabilities of its slice, it is reasonable that such QoE control is exposed by the P&P as well.

In this context, the role of a dedicated QoE optimizer P&P plugin is to enable the communication between the vertical and the slice optimization engine within the SliceNet platform. Figure 23 shows an overview of the P&P control and the slice QoE optimizer instances for a slice, and their interactions with the other elements of the SliceNet architecture. In a nutshell, the service/slice orchestration sub-plane is responsible for the set-up and deployment of the P&P control and the slice QoE optimizer. These two modules interact following the microservice approach described in this deliverable. The One-Stop API (OSA) is the entry point for the vertical (and other consumers) to obtain the contracted degree of control of the slice. Finally, the monitoring and cognition sub-planes provide the slice QoE optimizer with the tools to achieve an optimized slice performance.

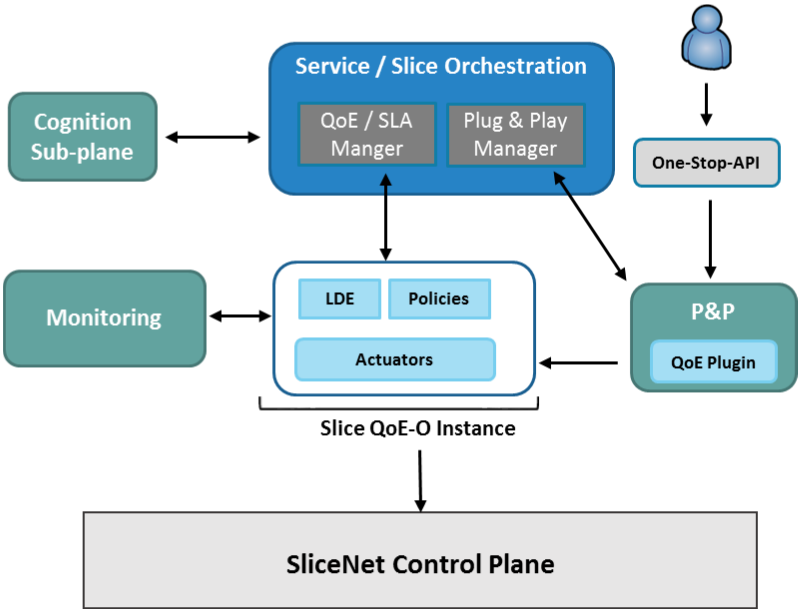


Figure 23 Slice Plug & Play and QoE optimization overview.

Once, deployed and activated, the QoE plugin exposes a set of operations that can be invoked through the slice view provided by the P&P control instance northbound interface. Mainly, two types of functions have been defined to be exposed and implemented by the plugin. First, the configuration functions allow the P&P control and management to set the proper configuration of the plugin (as previously defined in 4.3.3). As an example, the slice identifier and the location of the control plane resources to be invoked by the plugin can be set by these kinds of functions. Second, a set of operational functions implement the actual QoE-based slice control. Such functions can be invoked by the P&P consumers, i.e. the vertical or possibly other control functions in the P&P instance. The function that exemplifies this operational control in this version of the slice QoE optimizer plugin, is the QoE feedback provided by the vertical. Annex C contains the specification of the API exposed by the QoE plugin.

Figure 24 provides a view of the internals of the QoE plugin and its interactions with other modules of the SliceNet system. The REST API containing the operational and configuration functionalities is exposed through the P&P control to the One Stop API (and other potential P&P consumers), respectively. The QoE control functions invoked by the consumers trigger the slice QoE optimization process, which is requested to the slice QoE optimizer module of SliceNet. Finally, the operational logic needed by the plugin is implemented in the core of the plugin, which also contains the model of the information to be used and exchanged.

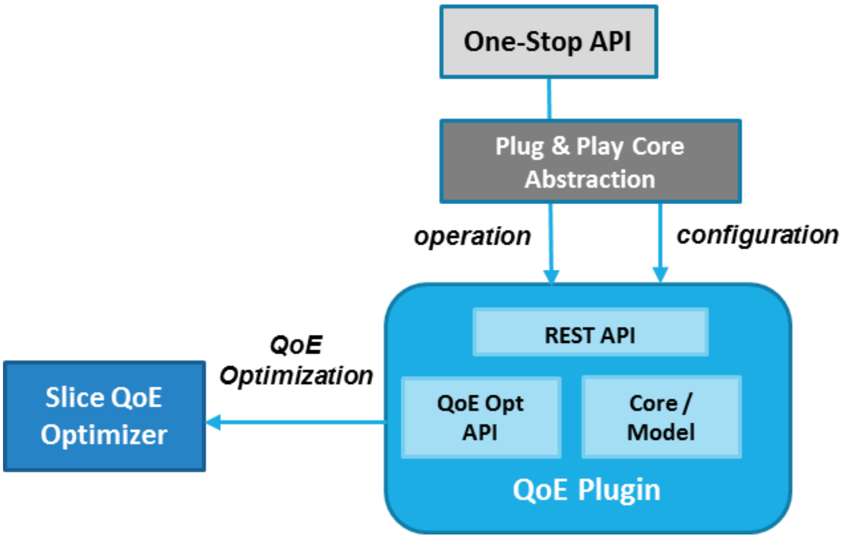


Figure 24 QoE Optimizer plugin general architecture and interfaces.

The operation of the QoE optimizer plugin is illustrated in Figure 25. Therein, the vertical uses the One-Stop API to report a QoE feedback to the Plug & Play (steps 1 and 2). This is automatically sent to the plugin that contacts the QoE Optimizer instance associated to the slice (step 3). If the reported QoE requires it, the QoE Optimizer performs the actions needed to improve the QoE of the users of the slice (steps 4 and 5). To do this, the slice QoE optimizer is fed with the inputs of both the cognition and the monitoring sub-planes.

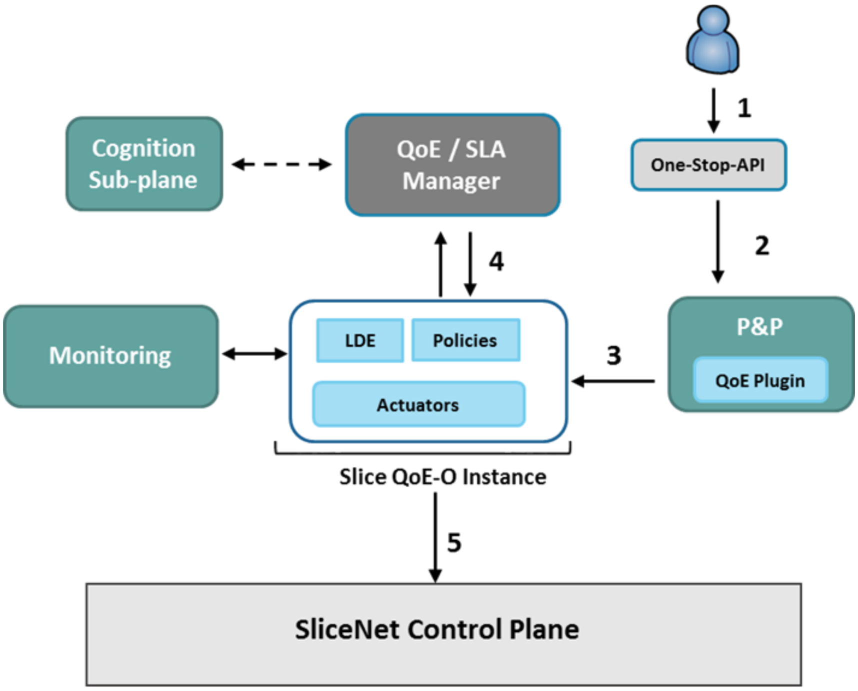


Figure 25 Plug & Play based Slice QoE optimization process.

4.4.1 QoE Optimizer as an integrated Plug & Play control function

The P&P and the QoE optimizer entities, as they have been defined in the SliceNet project, have a number of similarities. In particular, both entities are aimed at achieving a per-slice control and operation, and both of them are, architecturally speaking, placed between the SliceNet control and management planes. For this reason, an enhanced version of the Plug & Play, natively integrating slice optimization functionalities is envisioned.

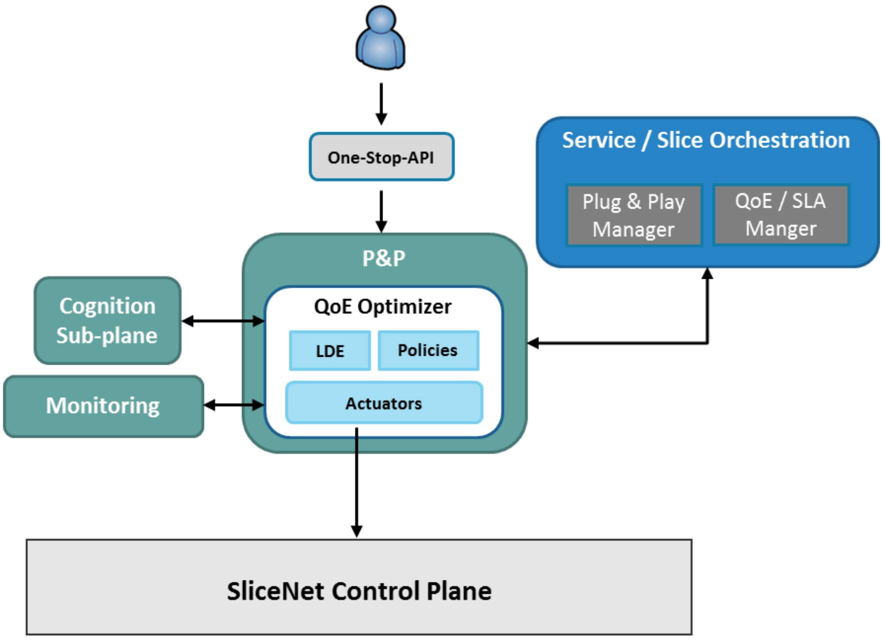


Figure 26 Slice QoE Optimization enabled Plug & Play control.

Figure 26 illustrates the general architecture of an enhanced P&P control instance, zooming in the slice optimization service. From a functional perspective, this new approach has not a big impact on the Plug & Play control architecture that has been described along this deliverable. Neither has it big implications on the current implementation of the Slice QoE optimizer module. Contrariwise, this new approach reduces the communication overhead between the P&P control and the Slice QoE optimizer and simplifies the per-slice control software infrastructure. As a matter of fact, the abovementioned communication interface between the QoE plugin and the Slice QoE Optimizer is not needed anymore in this model.

4.5 Plug & Play deployment models

Starting from available definitions from standardization bodies, such as 3GPP [27], SliceNet has identified the business roles pertinent and relevant for the delivery of end-to-end network slices to vertical industries [16].

- Network Service Provider (NSP): it is the entity that manages, exposes and monetizes network infrastructure resources that can be utilised to provide slices. It provides network slice as a service, and slices offered by an NSP span across its end-to-end infrastructure;
- Digital Service Provider (DSP): it is the entity that manages, exposes and monetizes digital services that are deployed on top of network slices as provided by one or more NSPs;
- Digital Service Customer (DSC): it represents the vertical entity itself, as ultimate consumer of digital services.

More than that, SliceNet has also defined how these business roles and the proposed logical architecture [1] map to both single and multi-domain scenario, identifying high-level interactions among the involved actors as well. In particular, SliceNet allows the same actor to implement more than one business role, inline with heterogeneous categories of actual Service Providers in the market. For this, the cases of standalone NSP, standalone DSP, and combined NSP and DSP are defined [16].

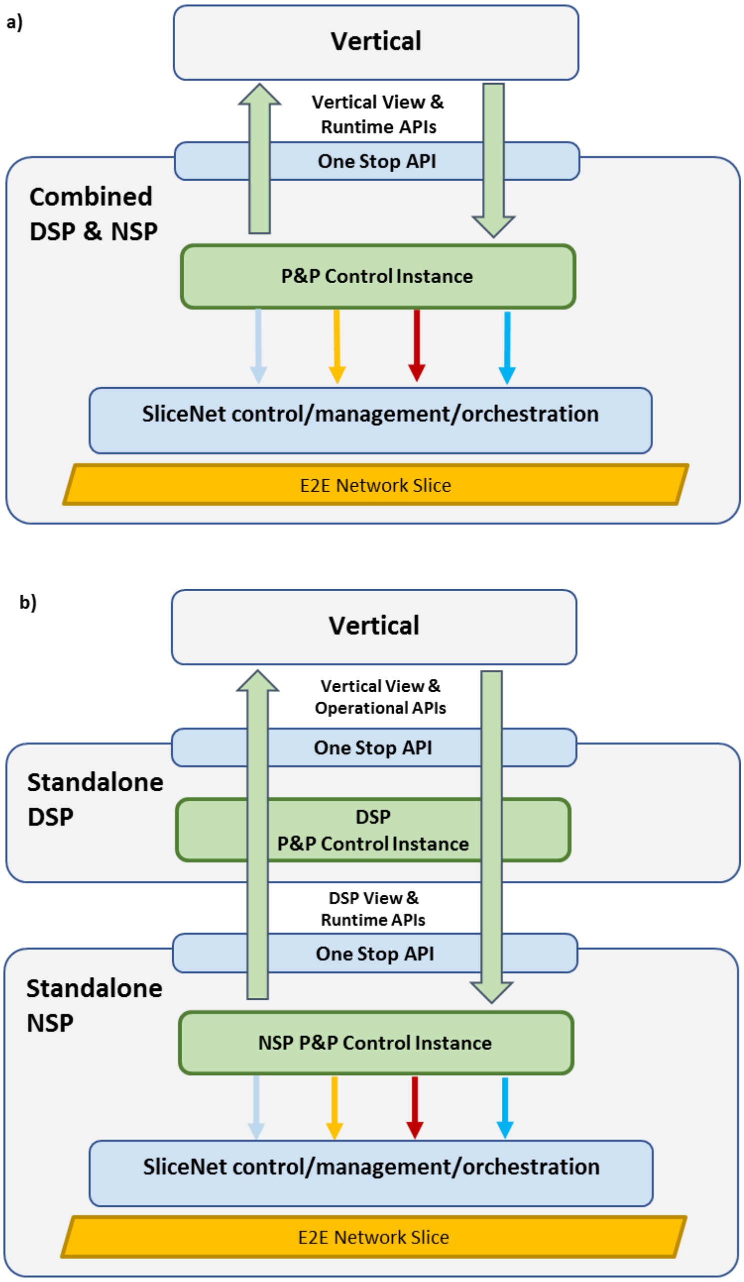


Figure 27: Plug & Play single-domain deployment: a) combined NSP & DSP, b) standalone NSP & DSP

In this context, the P&P control, as it has been architected, is a common framework that can be deployed and used as customization enabler at both NSP and DSP levels. This means that the P&P control framework approach described in the previous sections can be adopted for any slice consumer to slice provider interaction, thus covering both the DSP-to-vertical as well as the NSP-to-DSP exposure. In addition, the P&P control is also still valid with the same design approach in both single and multi-domain scenarios. Indeed, as anticipated in section 4.1.2, the P&P control can fit different deployment models according to the availability of proper control functions and plugins able to deal with different types of P&P to P&P interactions. In particular, four deployment models have been defined for the P&P control according to the different options available for single or multi-domain cases and standalone or combined NSP and DSP, as follows:

- single-domain, combined NSP and DSP deployment
- single-domain, standalone NSP and DSP deployment
- multi-domain, one combined NSP and DSP, one (or more) NSP standalone
- multi-domain, standalone NSPs and DSP deployment

The idea of describing these models in this document is to highlight how the P&P control framework is versatile and extensible enough to be adopted in different scenarios. In practice, the P&P workflows and mechanisms described in section 4.2 are applicable to these four deployment models as well. However, the details of each of these models in terms of interactions, overall orchestration workflows and information exchanged among actors and domains are being discussed in the context of WP7 to properly map each of the three SliceNet use cases (SmartGrid, SmartCity, eHealth) with respect to the business roles.

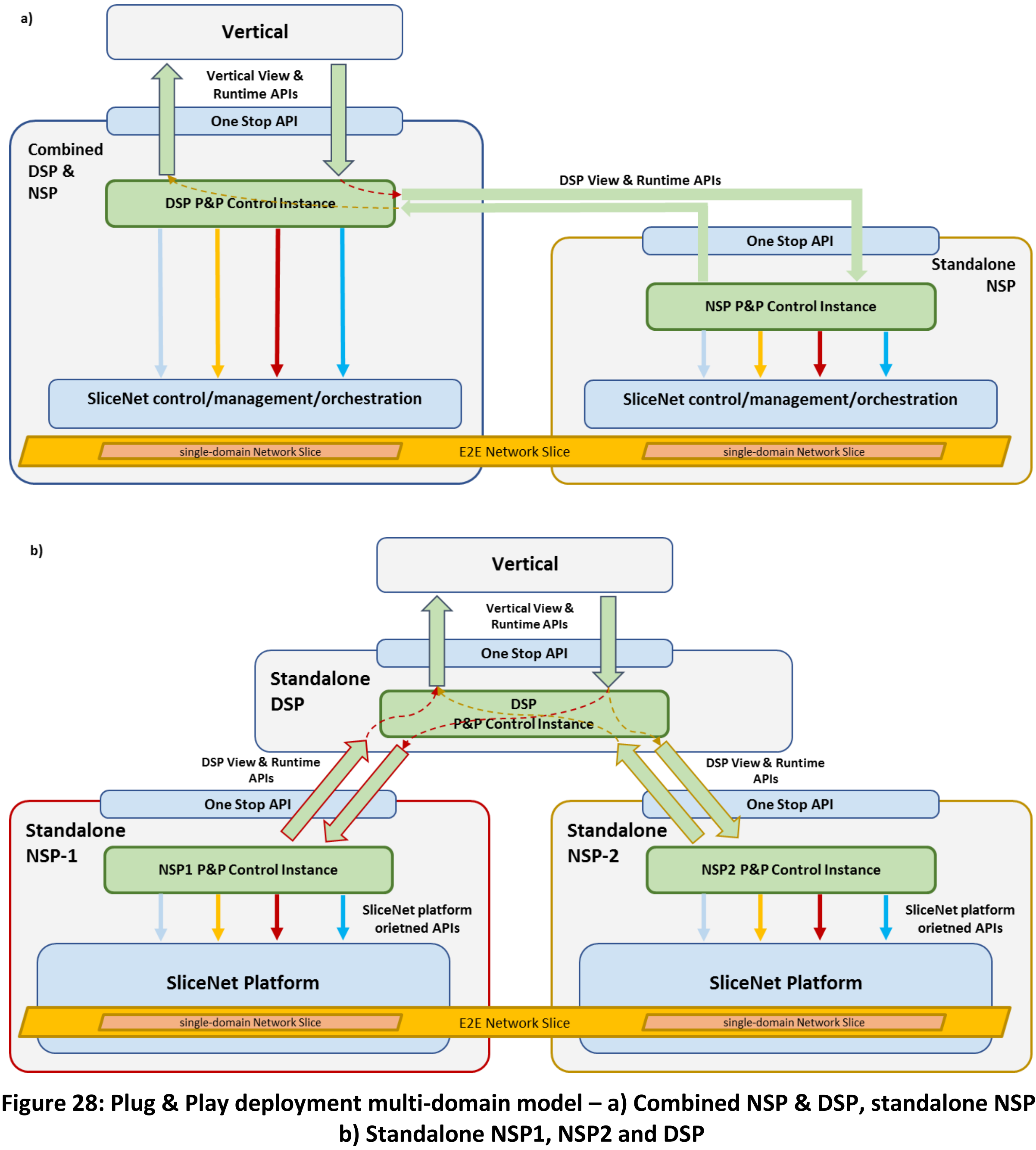


Figure 28: Plug & Play deployment multi-domain model – a) Combined NSP & DSP, standalone NSP, b) Standalone NSP1, NSP2 and DSP

The single-domain models are shown in Figure 27. The combined NSP and DSP deployment is the one in Figure 27a and it represents the simplest model for the adoption of the P&P control framework. A single-domain network slice is offered to the vertical by a unique actor implementing both NSP and DSP roles, requiring a single P&P control instance. This is the deployment model currently envisaged for the SmartGrid use case. On the other hand, the single-domain, standalone NSP and DSP is shown in Figure 27b and it represents a little more complex single-domain scenario where two separate actors interact between each other to provide digital services to a vertical. In this case, two independent P&P control instances may be required to properly abstract, towards the vertical, the view of the network slice and the related offered runtime control operations on top of it. Indeed, the NSP P&P control instance is intended to provide a tailored slice control exposure for the DSP, which may want to leverage to fulfil its own runtime control logics by itself. On the other hand, the DSP P&P control instance may be required to further abstract and limit that slice exposure, as the DSP may want to not expose to the vertical the full set of P&P capabilities offered by the NSP. Currently, this is the deployment envisaged for eHealth use case.

For what concerns multi-domain deployment models, they are depicted in Figure 28. In the picture, a simple case of two domains is represented. However, the same model and considerations described below are valid for potential cases with more than two domains involved. In the case of one combined NSP and DSP with one (or more) standalone NSPs, shown in Figure 28a, two main actors cooperate to provide an end-to-end network slice to the vertical, which only has business relations with the combined NSP and DSP. The slice spans across at least a couple of administrative domains. Here, the combined NSP and DSP is the one coordinating the multi-domain interactions and provisioning, and two P&P control instances are envisaged. The NSP P&P on the right is offered by the standalone NSP to the combined NSP and DSP on the left. It is possible that this P&P, being offered to a different administrative (and possibly competitive) entity, exposes very limited runtime control capabilities. The DSP P&P control on the left of Figure 28a is the one exposed to the vertical, that leverages on the DSP view and runtime APIs offered by the NSP P&P to offer a comprehensive set of P&P capabilities to the vertical. It combines the cross domain P&P capabilities and slice view while customizing the vertical view and runtime control APIs according to the vertical business logics and requirements. On the other hand, in the case of standalone NSPs and DSP deployment, shown in Figure 28b, a DSP trades an end-to-end network slice. For what concerns P&P features, each NSP exposes customized view and runtime APIs to the DSP through dedicated NSP P&P control instances. At the DSP side, a further P&P control instance can be deployed to decouple the P&P capabilities of the NSPs and above all expose to the vertical a customized slice view and control capabilities according to its requirements. It is relevant to highlight that the DSP itself (e.g. within its slice and service orchestration components) could leverage the P&P control APIs offered by each NSP to apply its own runtime control logics.

5 Plug & Play control software prototype

The P&P control framework software prototype has been developed from scratch in SliceNet, in the context of T4.1 activities, and it provides the full set of P&P features and operations according to the high level architecture, procedures, information models and APIs defined in section 4. Most of the P&P software prototype components have been developed as Pyhton3 projects, even if the microservices approach enable the usage of different languages and frameworks (e.g Java) at least at the P&P plugin and control function level.

At the time of writing, the whole P&P control framework software has to be considered as closed source code. Nextworks, as main contributor of the P&P core functionalities, plugins and control functions, together with other T4.1 contributors is investigating the possibility to release the whole P&P control framework software as open source, and integrate it as an additional tool of the company's slice and NFV orchestration suite [28][29]. For the moment, the P&P control framework source code is available in a Nextworks' internal git repository, that has been also used during the development phase.

The following sections provide a brief highlight on the P&P control framework prototype mainly in terms of: i) reference frameworks used for containerization and orchestration of the P&P microservices, ii) P&P microservices software, iii) installation and operation guidelines, iv) preliminary P&P integration and validation in the context of the SmartGrid use case.

5.1 Reference frameworks

This section describes how the P&P control framework microservice principle and architecture approach described in the previous sections have been applied to develop a software prototype able to offer a per slice flexible and highly customizable control instance for verticals that can include monitoring and management features.

The base idea is to conceive each P&P control instance, which is modular by definition, as a "containerized" application, in which a single functionality module (e.g. a plugin or control function microservice) is packaged in a single software container. Following this approach, a P&P control instance can be considered as a set of containers interacting together, in order to provide all of the features the vertical can take advantage of for controlling, monitoring and managing its own slice. The composition of this set is, in general, variable (only P&P core abstraction is always present, see Section 5.2) and is tailored to the vertical requirements.

With this in mind, each microservice building the P&P control instance is encapsulated in a container by using proper tools for software containerization, while a container orchestrator is in charge of managing the lifecycle of the consequent set of containers. Aiming to take benefit of available open source tools, P&P leverages on Kubernetes [18]as the container orchestrator while containerization technology is provided by the well-known and widespread Docker Engine [17]. Both of them are Open Source and usable under Apache 2.0 licence, and can be considered as de-facto standard for containerization and orchestration of microservice based applications.

Docker Engine is a multi-layer application consisting of a server, the Docker daemon (*dockerd*), working at so-called layer 0, that is in charge of the containers lifecycle. A user (Kubernetes in the P&P context) can access all Docker Engine feature through its command line interface. On top of the daemon server, Docker Engine offers a set of REST API (layer 1) and a command line interface (CLI, layer 2): the *docker* command (see Section 5.4.2).

To enable a user to build a customized container environment, Docker defines and offers a set of objects, called Docker Objects. Here, *Containers* and *Images* are the more relevant Docker Objects in the P&P context:

- **container**: it is defined as a runnable instance of an image and it can be created, started, stopped, moved, updated and destroyed by using Docker Engine command line interface
- **image:** it is a kind of template, including a set of instruction needed for creating containers. Docker offers a set of base image (e.g. Ubuntu) that a user can augment in order to create its own custom image (see Section 5.4.2). Once created, an image is stored on a Docker Registry that can be local to the user container environment or remote. Docker offers a set of public trusted repositories to be possibly used.

While Docker allows to deploy the P&P microservices as containerized applications through its features described above, Kubernetes is the tool that the P&P control framework leverages to deploy, manage and orchestrate these containerized applications in a very flexible and scalable way. One of the key concepts of Kubernetes that are relevant to P&P is the **POD**. Kubernetes consider a POD as an abstraction that represents a set (one or more) of containers. It also represents the minimum "runnable" unit within Kubernetes. This means that Kubernetes orchestrates containers in term of PODs: to run a single container, it needs to run a POD including that (single) container. Another key relevant Kubernetes concept is the **Service**. It is another Kubernetes abstraction that defines a logical set of PODs and the set of policies to access it.

For the P&P control, the idea is to realize the dedicated per-slice control instance (from now on referred as P&P instance) as a single-POD Service. In practice, for each P&P instance, a Kubernetes Service exposing the slice control interface towards the One-Stop-API framework, consists of a single POD that abstracts the set of containers providing the customized slice view and runtime control APIs for the vertical.

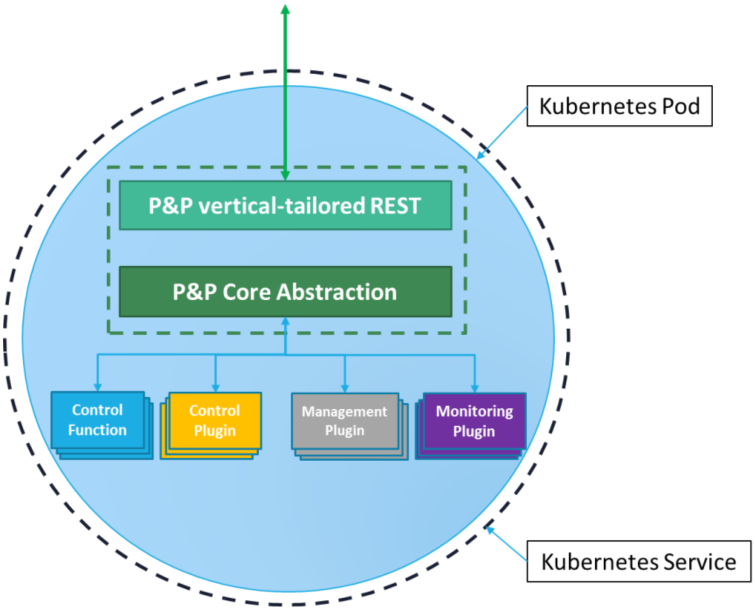


Figure 29 P&P control instance deployed as a single Kubernetes Service

Figure 29 shows the general composition of a given P&P instance. The set of containers, represented as boxes, consists of plugins and control functions providing customized control, management and monitoring features, and an additional container (green dotted rectangle) that contains the P&P core abstraction and vertical-tailored APIs functionalities, including: slice control exposure handling, APIs exposition, plugin handling, as described in Section 5.2.

The blue arrows between the containers in Figure 29 represents the REST communication interfaces, in line with the definitions in section 4.3. The POD representing the abstraction of the containers is depicted as a light blue circle, while the dashed circle enclosing it is a representation for the Kubernetes Service. The Service exposes the P&P vertical-tailored REST APIs towards the One-Stop-

API framework (green arrow). It is relevant to highlight that authentication and role-based access features to P&P control instances are intended to be provided by means of the One-Stop-API framework [30].

In general, from the Kubernetes perspective, a Service relies on a cluster of nodes for deployment of the different PODs and containers. A node is a worker machine or server containing the running PODs. The minimum Kubernetes cluster consist at least of two nodes, one Master, in which all Kubernetes control components run (APIs server, Scheduler, Dashboard, etc) and one (at least) “component” node in which PODs and containers are deployed. Therefore, by default, containers cannot run inside the Master node.

P&P as clustered application, with PODs and containers running is multi-node environments, is therefore an option that Kubernetes natively offers. However, for the initial validation P&P deployments, a single Kubernetes node approach has been followed. In particular the Master node can be configured to be also a worker node by using the Kubernetes command ***taint***. In this way, the P&P control framework relies on a single-node cluster in which the single-node is, at the same time, both master and worker. However, this solution does not compromise any possibility of adding new worker node in the future to fulfil specific additional P&P distributed deployment requirements, that may arise as part of the evolution of SmartGrid, SmartCity and eHealth use cases deployment scenarios.

The single-node Kubernetes cluster can be a physical or a virtual machine as shown in Figure 30. Services, PODs and containers are visible inside the node and represents multiple instance of P&P (one per slice) exposing their REST APIs towards the One-Stop-API layer. The figure also shows a dashed box that represents the P&P Manager [23], as the responsible for the P&P instance lifecycle management (create, update, delete) and configuration. Such procedures are currently manually executed by directly accessing the Kubernetes interface, as explained in Section 5.4.2. The P&P Manager will allow to fully automatize these procedures and integrate them with slice and service orchestration workflows.

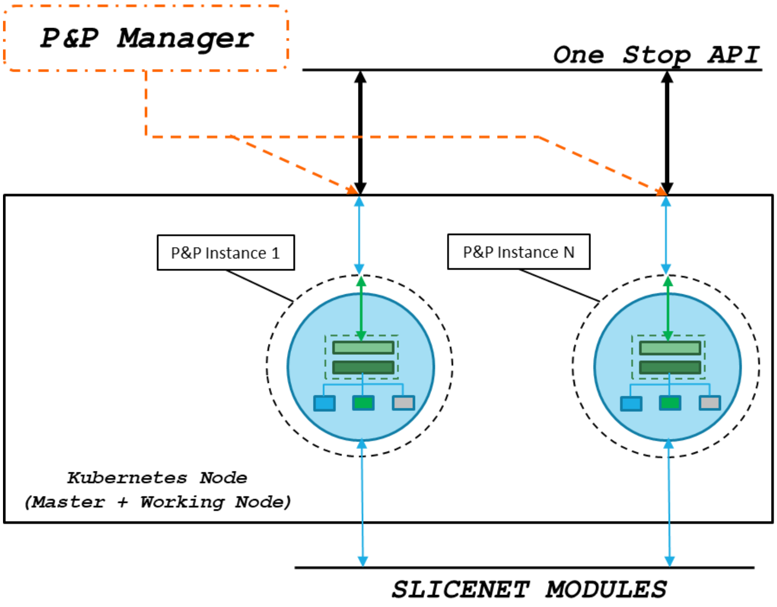


Figure 30 Kubernetes single-node cluster

In summary, the combined use of Docker containers and Kubernetes orchestration has at least two major advantages. The first is enabled by the container technology itself: as each P&P plugin and control function running in a container provides a well-defined and atomic control feature to the vertical, it is straightforward to expose a per-slice customized set of services by combining the proper set of containers. The second one is enabled by Kubernetes, as it allows updated of the set of containers within a POD at runtime, without any kind of service outage. Therefore, as soon as the vertical requirements for a given slice change in terms of customized exposure, the P&P can dynamically adapt in a fully transparent way for the vertical.

5.2 Plug & Play core abstraction prototype

As described in Section 4.1.1.2, the P&P core abstraction microservice consists of a set of internal modules developed to handle the connected plugins and control functions while abstracting their own offered services through a proper REST Interface, according to the customized slice exposure view based on the information models defined in section 4.3.1. From a software implementation perspective, this microservice also embeds the vertical-tailored REST API microservice functionalities.

All of P&P core abstraction modules are developed in Python3 and, in general, they are validated for Python v3.5. Along with the standard Python3 facilities, in order to provide all functionalities required, with reference to some of the internal features are based on specific framework and tools:

- **REST APIs**: they are implemented on top of Tornado [31], a scalable web framework that offers all facilities to expose a REST interfaces and handle related queries. It offers also a mechanism for websockets management.
- **Engine and Store:** are modules written in standard Python3. In particular, the store consists of a collection of Python3 dictionaries: due to this, the information stored in the core are destroyed once the core is destroyed or rebooted.
- **Core REST client:** is based on Requests HTTP library [32], that simplify, from the development perspective, the management of the HTTP requests and replies.
- **Core Parser:** a module supporting two API definition format: Swagger 2.0 and OpenAPI 3.0. On one hand, the parsing of Swagger 2.0 model is based on a module, compatible with Python3, called Swagger-Parser [33]. On the other hand, OpenAPI 3.0 support is currently partial due to the lack of available and stable off-the-shelf Python3 packages. The current OpenAPI 3.0 parsing features are developed from scratch in standard Python3 and it is envisaged they will be consolidated and further refined as part of the SliceNet integration activities.

5.3 Plug & Play plugins and control functions prototypes

A set of P&P control functions and plugins have been developed, mostly in support of the preliminary P&P integration and validation in the context of the SmartGrid use case. In particular, three software prototypes have been implemented: the QoE optimizer control function, the User Equipment control plugin, the QoS monitoring control functions.

The following subsections provide a brief highlight for each of these software prototypes.

5.3.1 QoE optimizer control function

The software release of the Slice QoE Optimizer control function consists of a software module that has been developed in Java8 and released as a Docker container ready to be deployed as a Plug & Play service. From a software architecture perspective, the QoE Optimizer control function has been split in several sub-modules (Figure 31):

- The plugin API sub-module implements the REST interface that enables the communication with the P&P core abstraction and thus, exposes the configuration and operation functions provided by the control function itself. This sub-module also implements a model of the QoE optimizer that is then used by the core part of the control function to get the inputs received through the API. The OpenAPI 3.0 has been used to define the REST interface and the model, which are detailed in Annex C. The Java Spring library [34] has been used to implement this sub-module.

- The Optimizer sub-module is responsible for triggering the optimization process when a QoE feedback is received from the vertical. To do this, the Optimizer sub-module is composed of a Policy Framework manager, which is able to store policies, and an Actuation manager that is able to apply the actions associated to such policies. This current release of the optimization core is intended to be a first version of the whole slice QoE optimization service integrated in the P&P as described in section 4.4.1
- To implement the abovementioned actuation, the QoE plugin has been equipped with a sub-module that contacts the SliceNet control plane. The goal of this sub-module is to enable the Actuation manager of the Optimizer sub-module to execute actions over the slice infrastructure through the SliceNet control plane. Hence, the control plane sub-module is able to call the control functions exposed by the SliceNet control plane. To do this, it contacts the CPSR [26] to request the different types of control functions that are available (i.e. QoS control, Network Functions configuration, etc). In this way, upon a QoE feedback from the vertical, the Policy Framework manager decides if an actuation is needed and which one has to be performed. The Actuation manager contacts the control plane sub-module to request the action to be taken, and the control plane sub-module contacts in turn the SliceNet control plane, which is the one that interacts with the slice infrastructure.

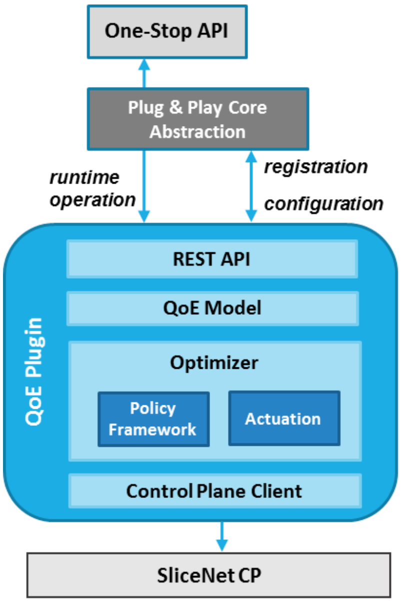


Figure 31 Slice QoE Optimizer software architecture

5.3.2 UE control plugin

The User Equipment (UE) control plugin aims at providing access to UE control and management features exposed by the SliceNet platform. In particular, the main goal of this plugin is to abstract these features for vertical purposes, thus restricting the set of information and resources accessible. The UE control plugin is indeed conceived to implement the requiring hooking to those SliceNet control plane technology agnostic APIs that gives access to UE related information on the one hand, and that allows to attach (or detach) UEs from existing network slices on the other.

The UE control plugin software architecture is shown in Figure 32, and it is compliant with the common functional split for plugins and control functions defined in section 4.1.1.1. The main control operations exposed by the plugin through its REST APIs are:

- Associate a new UE to the slice instance
- Remove a UE from the slice instance
- Retrieve status information of a UE in the slice instance, given its International Mobile Subscriber Identity (IMSI), according to the model defined in Table 26

Table 26 UE status information model

Identifier	Cardinality	Description
imsi	1	It is the unique international identifier of UE subscriber identity.
ip	1	It is the IP of the UE in the (sliced) mobile network. It could be subject to Network Address Translation (NAT)
location	1	It identifies the (RAN) location where the UE is attached to the (sliced) mobile network
status	1	It is the status of the UE in the (sliced) mobile network. It can be: “ATTACHED”, “DETACHED”.

The detailed OpenAPI specification of the whole set of REST APIs exposed by the UE control plugin is reported in Annex D. The Control Plane Client allows to interact with the UE control and management technology agnostic APIs in the SliceNet control plane [26], and it is handled by the Abstraction Engine which hides the details, resources and data models of such APIs towards the P&P core abstraction and the One-Stop-API.

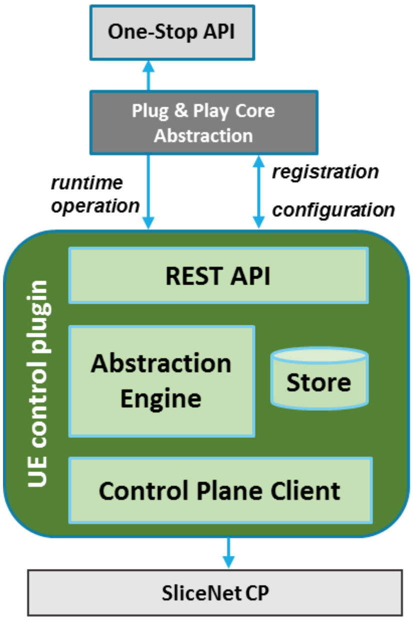
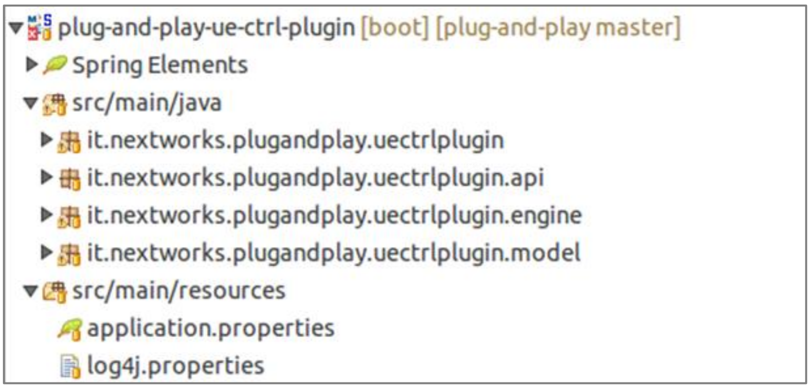


Figure 32 UE control plugin software architecture

From a software perspective, the UE control plugin is a self-contained Apache Maven project [35], and it has been developed from scratch as a Java Spring application [34]. The source code is organized in several packages, as shown in Figure 33. Each package maps to a functional box in Figure 32, with the Control Plane Client that is actually embedded in the *engine* package, and the *model* package providing the structure for objects managed by the UE control plugin and then maintained in the Store component.



```
▼ plug-and-play-ue-ctrl-plugin [boot] [plug-and-play master]
  ▶ Spring Elements
  ▼ src/main/java
    ▶ it.nextworks.plugandplay.uectrlplugin
    ▶ it.nextworks.plugandplay.uectrlplugin.api
    ▶ it.nextworks.plugandplay.uectrlplugin.engine
    ▶ it.nextworks.plugandplay.uectrlplugin.model
  ▼ src/main/resources
      application.properties
      log4j.properties
```

Figure 33 UE control plugins source code structure

5.3.3 QoS monitoring control function

The QoS monitoring control function aims at producing slice-level performance statistics and KPIs by elaborating raw monitoring data offered by the SliceNet monitoring sub-plane. In particular, the control function computes the QoS in terms of throughput measurements at both slice and UE levels. These latter refer to QoS metrics of UEs attached (i.e. belonging) to the given slice the P&P control instance is supporting.

Figure 34 shows QoS monitoring control function internals, and it follows the P&P plugins and control functions common approach described in section 4.1.1.1. It exposes a REST Northbound interface, defined in Swagger 2.0 format (See Annex E) towards the P&P core abstraction, while at its southbound, a REST client enables the collection of monitoring information from the underlying SliceNet monitoring sub-plane components, currently under development in WP6. A Store module maintain essential information such generic slice instance information and the configuration data required to interact with the SliceNet monitoring sub-plane components.

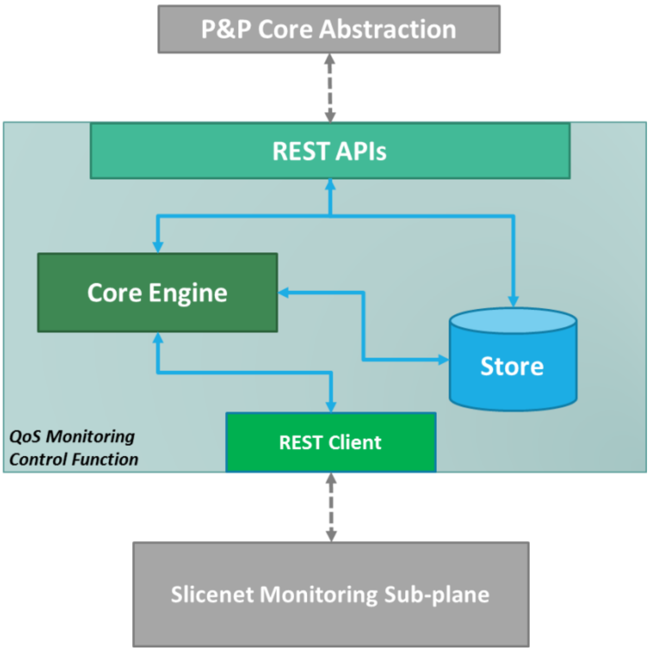


Figure 34 QoS Plugin software architecture

The Core module takes car of: i) collecting the raw data retrieved from SliceNet Monitoring sub-plane through queries performed by means of the REST client, ii) elaborating and aggregating the raw data, iii) and producing a set of new slice and UE level metrics to be exposed to the vertical through the

One-Stop-API by means of the control function REST APIs (in this case offered through explicit GET queries).

In particular this QoS monitoring control function consumes performance statistics offered by the SliceNet monitoring sub-plane and referring to the RAN network segment, and produces, by means of dedicated aggregation rules, a set of aggregated and instantaneous metrics expressing a measure of the throughput. Table 27 shows the set of throughput metrics generated by the QoS monitoring control function at slice level.

Table 27 Slice QoS information

Name	Type	Description
slice_id	Identifier	Slice identifier
avg_slice_dl_throughput	Aggregated	Average Slice download throughput calculated from time series build from RAN information
avg_slice_ul_throughput	Aggregated	Average Slice upload throughput calculated from time series build from RAN information
slice_dl_throughput	Instant/Aggregated	Slice download throughput calculated as sum of single UE’s instant download throughput
slice_ul_throughput	Instant/Aggregated	Slice upload throughput calculated as sum of single UE’s instant upload throughput

A similar set of performance metrics are produced by the QoS monitoring control function for the UEs attached to the given slice, as listed in Table 28.

Table 28 UE QoS information

Name	Type	Description
imsi	Identifier	UE unique identifier
avg_dl_througput	Aggregated	Average UE download throughput calculated from time series build from RAN information
avg_ul_througput	Aggregated	Average UE upload throughput calculated from time series build from RAN information
dl_througput	Instant	UE download throughput calculated from RAN instant information
ul_througput	Instant	UE upload throughput calculated from RAN instant information

Moreover, the QoS monitoring control function produces a set of additional performance metrics related to the data flows handled by each UE, as defined in Table 29 below.

Table 29 Data flow statistics per UE

Name	Type	Description
flow_id	Identifier	Identifier of the data flow
imsi	Identifier	UE unique identifier as registered in the 4G/LTE network slice
byte_count_ul	Instant	Number of uplink transmitted bytes (one entry per direction: DL/UL)
byte_count_dl	Instant	Number of uplink transmitted bytes (one entry per direction: DL/UL)
packet_count_ul	Instant	Number of downlink transmitted packets
packet_count_dl	Instant	Number of downlink transmitted packets

The QoS Plugin follows a similar software development approach to the P&P core abstraction (see section 5.1.1). The northbound REST APIs are implemented on top of *Tornado Framework [31]*, while the southbound REST client uses structures and functions provided by *Requests HTTP Library [32]*. The other functional components, i.e. Core and Store have been developed as Python3 modules.

5.4 Release format and operation guidelines

The main goal of this section is to describe the requirements that need to be satisfied in order to build and deploy a P&P control instance.

5.4.1 Dependencies and pre-requisites

As described in the previous sections, the P&P is conceived as a containerized application and each instance is deployed as a Kubernetes Service running on a Kubernetes node. Therefore, the machine, either virtual or physical, in which the P&P instances run, shall meet two requirements summarized in Table 30.

Table 30 P&P Pre-requisites

Tool Name and Version	Description
Kubernetes 1.11.3	Kubernetes is the container orchestrator that handle each P&P instance lifecycle
Docker Engine 17.03.2-ce	It is the platform that provides the containerization technology and tools to build, deploy and run containers.

In addition to the previous requirements, that can be considered as basic ones, there are also software dependencies that have to be satisfied to properly run the P&P containers. Table 31 below list the software dependencies related to the P&P core abstraction, as the main microservice and container common to all P&P instances.

Table 31 P&P software dependencies

Toll Name and Version	Description
Python 3.5	All P&P core modules developed, including the ones relying on particular packages, are compatible with Python version 3.5
Tornado 5.1	It is the Python web application framework used to implement the P&P REST API and the continuous monitoring based on websockets
Requests 2.19.1	It is the Python HTTP library used to implement the P&P core abstraction REST Client
Swagger-Parser 1.0.1	It is a Python module integrated in the P&P core abstraction parser used to parse the Swagger API model of the plugins

One of the main advantages in the approach followed with P&P is that containers are loosely coupled and they interact only through respective REST APIs. For that reason, there are no other common software dependencies as each P&P control function or plugin can follow its own approach in terms of programming language, libraries, frameworks and tools. In this way, it is possible to integrate also third party control functions and plugins that satisfy the following constraints:

- Developed in a programming language supported by Docker
- Docker image shall contain all required files, libraries, packages required for properly run the control function/plugin
- Provide a REST API model in Swagger 2.0 or OpenAPI 3.0 format [19], including all APIs exposed and offered

5.4.2 Installation and operation

As Kubernetes is a container orchestrator, the first step to deploy a P&P instance is to create the images for the containers: the P&P core abstraction (embedding the vertical-tailored APIs) and the set of control functions and plugins.

To create an image suitable for a container, Docker provides the command *build*. An example of its use is the following (Snippet 3):

```
docker build –t <image_name>:<version> -f <dockerfile> <context_path>
```

Snippet 3 Docker build command

where:

- **image_name** and **version** are respectively the name of the image and its version, e.g: *core:1.0.0*
- **dockerfile** is a docker configuration file that contains all information to build the image.
- **context_path** is the path of the Docker Context. Docker context is the set of file (needed for building the image) located in the specified path.

The Snippet 4 below shows the content of the Dockerfile used to build the P&P Core image.

```
FROM python:3.5-alpine
RUN pip install tornado
RUN pip install requests
RUN pip install swagger_parser
COPY core_ui.py .
COPY core_engine.py .
COPY core_store.py .
COPY core_rest_base_client.py .
COPY core_api_parser.py .
COPY core_exceptions.py .
EXPOSE 60001
EXPOSE 60002
CMD python3 core_ui.py
```

Snippet 4 Example of Dockerfile

Once container images have been created, it is possible to create a P&P instance as a Kubernetes service. Kubernetes offers several ways to create a containerized application and make it accessible but, in any case, such procedures have as result the creation of the service in two steps:

1. Creation of the PODs and containers running inside them
2. Creation of the service that define the access policy to the POD

In the context of P&P, each instance consists of a single-POD service. Such POD is created by defining and creating a Kubernetes Deployment Object. A Deployment can be defined in a proper YAML file, that specifies full information about the POD like the set of containers that should run inside the POD, how many PODs (i.e. replicas in clustered deployments) have to be created, etc.

Once the POD has been created, it is possible to make it accessible from outside the Kubernetes cluster by defining and creating a Kubernetes Service Object. Also the Service can be defined in a proper YAML file that specifies what the POD should be expose along with the access policy (e.g. protocols, ports).

Examples of Deployment and Service YAML files are shown in Snippet 5 and Snippet 6 respectively.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: plug-play-slice-efacec
spec:
  selector:
    matchLabels:
      app: plug-play-slice-efacec
  replicas: 1
  template:
    metadata:
      labels:
        app: plug-play-slice-efacec
    spec:
      containers:
      - name: core-container
        image: core:v1.1.0
        ports:
        - containerPort: 60001
        - containerPort: 60002
      - name: qos-container
        image: qos_plugin:v5.0.3
        ports:
        - containerPort: 65007
      - name: ue-ctrl
        image: ue-ctrl:v2.0.2
        ports:
        - containerPort: 65006
       - name: qoe-container
        image: slcntupc/qoe-plugin:latest
        ports:
        - containerPort: 65005
```

Snippet 5 Example of Kubernetes Deployment configuration file

```
apiVersion: v1
kind: Service
metadata:
  name: plug-play-slice-efacec
  labels:
    app: plug-play-slice-efacec
spec:
  type: NodePort
  ports:
  - name: http
    protocol: TCP
    port: 60002
    nodePort: 30001
  selector:
    app: plug-play-slice-efacec
```

Snippet 6 Example of Kubernetes Service configuration file

One way to create the Kubernetes Object described above is to use the command *apply* provided by *kubectl,* that is the Kubernetes command line tool, as following:

```
kubectl apply –f <object_file.yaml>
```

Snippet 7 Kubectl apply command

The command *apply* is very versatile: it creates the object and the configuration specified in <object_file.yaml>. In this way, to create a Kubernetes Service representing a P&P instance, the procedure is to execute *apply* command two consecutive times passing case by case the proper object files: one defining the Deployment first and one defining the Service object later.

The Kubernetes *apply* command also allows to perform updates of POD deployments. It is possible to add or remove one or more container from a given deployment by just adding or removing them from the related deployment YAML file. Then, by applying again the *apply* command with the updated YAML file, Kubernetes will transparently destroy the old deployment while creating the new one.

It is important to highlight that the Kubernetes Service is not affected from this PODs switching process. During the transient, the Kubernetes Service keeps offering the features provided by the old POD deployment while, once the switching process finished, it starts to expose the features provided by the new POD deployment.

Moreover, Kubernetes Services and POD deployments can be removed independently. To destroy these Kubernetes Objects, *kubectl* provides the *delete* command that can be used as follows:

```
kubectl delete <object_type> <object_name>
```

Snippet 8 Kubectl delete command

where *object_type* could be either a Service or a POD deployment, while the *object_name* is the one specified in the YAML files (See Snippet 5 and Snippet 6). For example, to remove the Deployment specified in Snippet 5, the following command can be executed:

```
kubectl delete deployment plug-play-slice-efacec
```

Snippet 9 Example of deployment deleting

All the Kubernetes coordination operations described above are currently performed manually. The final goal is to make the P&P instance lifecycle completely automated, in which Docker images and Kubernetes YAML will be generated dynamically for verticals and slice consumers, with deployment, update and removal of a single instance that will be performed by P&P Manager [23].

5.5 Preliminary integration and validation

The P&P control framework prototype described above has been integrated and validated in the context of the Smart Grid use case Phase 1 deployment in the Altice Labs testbed in Aveiro [20]. Here, a SmartGrid Protection & Control IED (Intelligent Electronic Device) real device is connected through a dedicated core (DECOR) network slice deployed over a 4G/LTE network infrastructure to a SCADA (Supervisory Control And Data Acquisition) application. The Runtime control and monitoring of such SmartGrid slice is exposed through a dedicated P&P instance to the vertical (i.e. Efacec). The overview of this preliminary P&P integration and validation scenario is depicted in Figure 35.

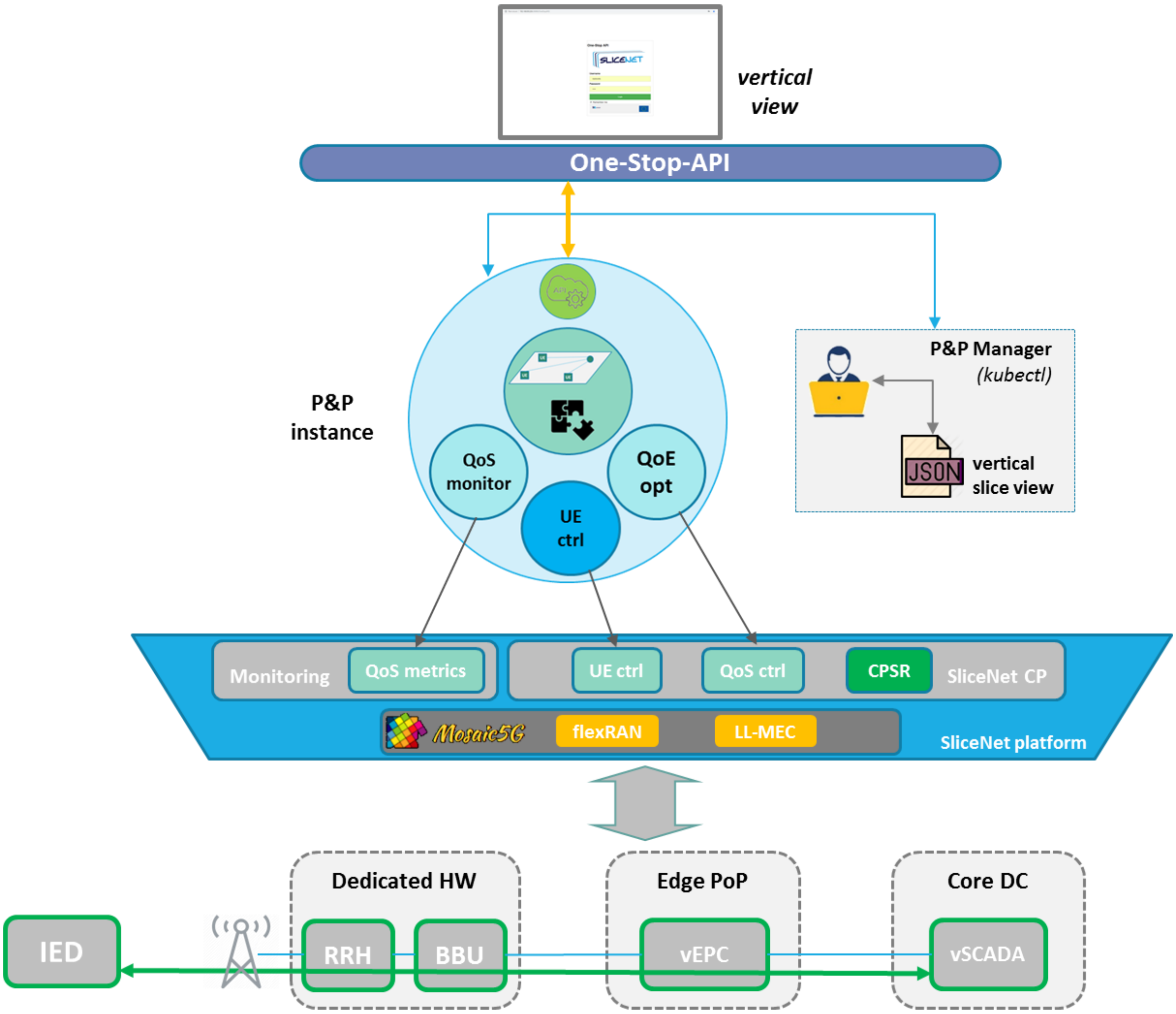


Figure 35 Plug & Play with One-Stop-API integration in the SmartGrid Use Case

The integration and validation setup was composed of four main layers:

I. the infrastructure, composed by a Protection & Control IED, a dedicated hardware running a Remote Radio Head Unit (RRH) and a Base Band Unite based on OpenAirInterface (OAI) RAN [36], an edge Point of Presence (PoP) running a virtualized Evolved Packet Core (vEPC) based on OAI CN [37], and a core Data Center hosting the SCADA application

II. a set of control tools implementing a preliminary prototype of the SliceNet platform, including the Mosaic5G FlexRAN controller [38] and the LL-MEC controller [39], as main SDN enabled solutions for controlling RAN and edge segments. On top of these, few SliceNet control plane services, including the CPSR for service discovery features (according to the service based architecture [2], a QoS control service and a UE control service exposes towards the P&P instance a set of technology agnostic APIs. In addition, a monitoring function called QoS metrics is also available to retrieve QoS performance statistics related to UEs and the RAN segment

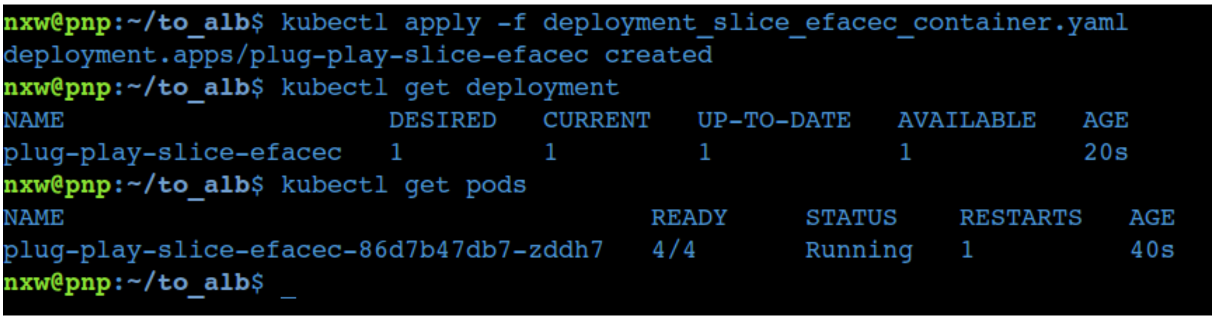
III. a P&P control instance dedicated for the SmartGrid network slice, composed of four microservices: the core abstraction, the QoE optimizer control function (see 5.3.1), the UE control plugin (see 5.3.2), the QoS monitoring control function (see 5.3.3) . As described above, the P&P Manager functionalities are manually provided by the Kubernetes command line facility

IV. a preliminary prototype of the One-Stop-API framework, including a Graphical User Interface (GUI) able to dynamically represent the vertical slice view and the related offered runtime control operations.

The main goal is of this preliminary integration was the validation of the P&P control framework prototype in the context of a network slice manually deployed and configured for the SmartGrid use case. In particular, the following

- P&P instance deployment & configuration
- Vertical-tailored smartGrid slice control exposure, in terms of:
  - IEDs/UEs status information retrieval
  - Collection of QoS statistics and slice performance
  - Slice expansion with new IEDs/UEs
  - Slice optimization triggered by vertical QoE feedback

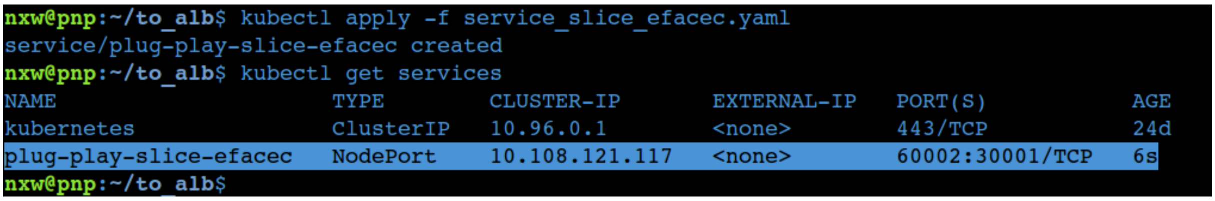
As said, it is assumed that the DECOR network slice represented by the green line in Figure 35 is pre-existent and manually configured, thus the IED is considered as already connected to the SCADA application. The first step for the P&P instance deployment consists in the creation of the POD with the containers running inside it. The result of this process, that follows the steps and procedures described in section 5.4.2, is shown in Figure 36. Here, the list of PODs created (only one, in this case) are shown with few details. Under the column READY is visible “4/4”, which means that 4 of the 4 deployed containers (P&P core abstraction and the three control functions/plugins) are running inside the POD.



```
nxw@pnp:~/to_alb$ kubectl apply -f deployment_slice_efacec_container.yaml
deployment.apps/plug-play-slice-efacec created
nxw@pnp:~/to_alb$ kubectl get deployment
NAME                     DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
plug-play-slice-efacec   1         1         1            1           20s
nxw@pnp:~/to_alb$ kubectl get pods
NAME                                      READY     STATUS    RESTARTS   AGE
plug-play-slice-efacec-86d7b47db7-zddh7   4/4       Running   1          40s
nxw@pnp:~/to_alb$ _
```

Figure 36 Kubernetes Deployment and Pod creation and list

Once the POD is up, it can be exposed as Kubernetes Service. After the Service has been created, it appears in the list of the running ones, as shown in Figure 37 below.



```
nxw@pnp:~/to_alb$ kubectl apply -f service_slice_efacec.yaml
service/plug-play-slice-efacec created
nxw@pnp:~/to_alb$ kubectl get services
NAME                     TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)           AGE
kubernetes               ClusterIP   10.96.0.1        <none>        443/TCP           24d
plug-play-slice-efacec   NodePort    10.108.121.117   <none>        60002:30001/TCP   6s
nxw@pnp:~/to_alb$
```

Figure 37 Kubernetes Service creation and list

At this stage, the Kubernetes Service implementing the P&P instance is up and running. The following step consists in configuring the new P&P instance by leveraging on the management APIs exposed at its northbound REST interface (see section 4.3.2.1). This configuration is a two-step process in which first the P&P instance is configured as per operation described in Table 21, and then it is provided with the vertical-tailored internal slice view. Once the P&P configuration is performed, the One-Stop-API GUI is able to retrieve from the P&P instance all the information needed to visualize the northbound slice view. Figure 38 shows the One-Stop-API GUI and the slice view topology, that is based on a high level exposure of slice components, including only IEDs/UEs logically interconnected with the SCADA application. This means that the smartGrid vertical do not requires (and thus did not request) any more granular and resource level view of its slice.

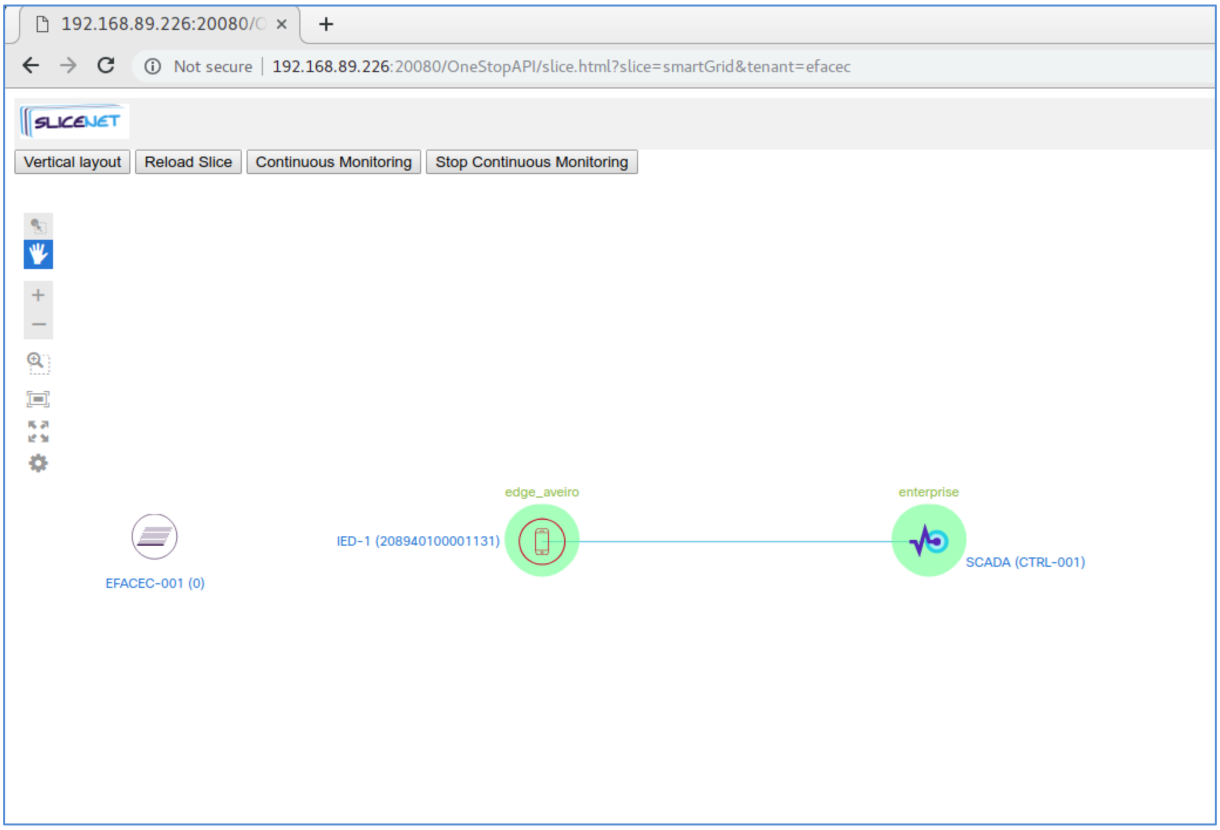
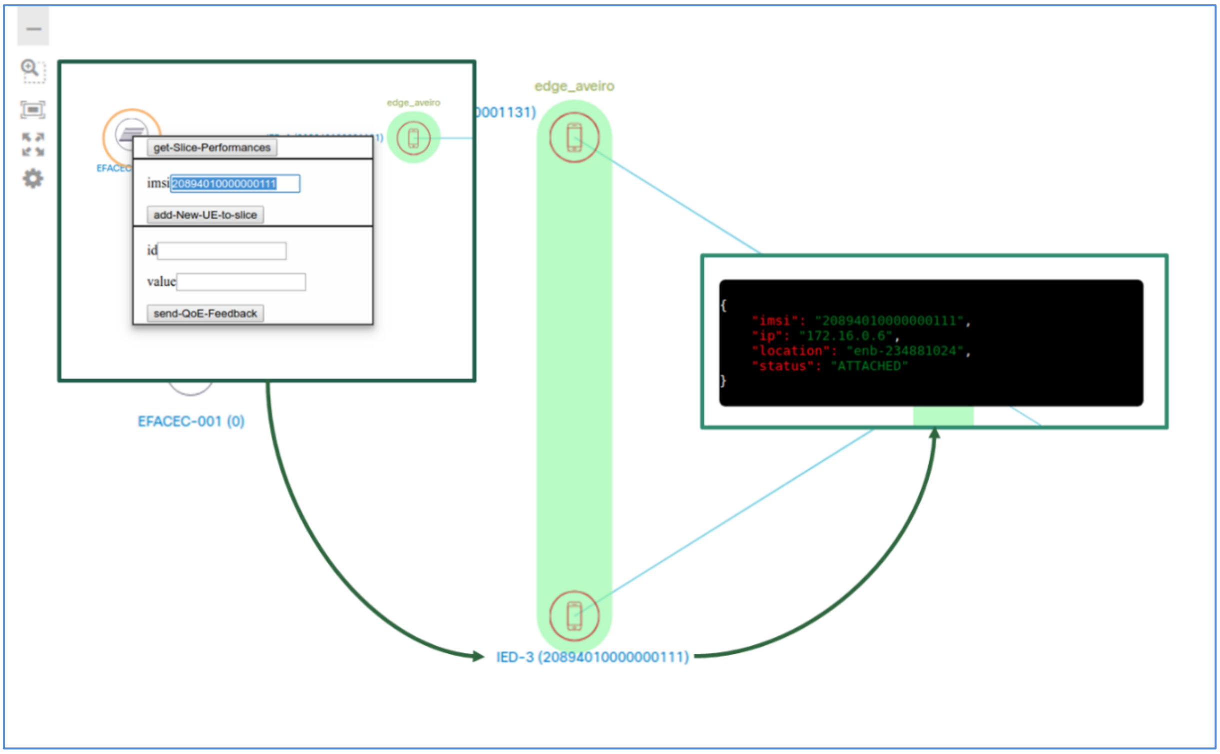


Figure 38 Slice view through One-Stop-API GUI

At this point, there is only one IED connected to the SCADA application. An additional icon is visible, on the left side of the GUI, representing . By clicking on this icon, as well as on any other slice view element icon, the One-Stop-API GUI lists the available runtime control operations that P&P control exposes. Therefore, from now on, the vertical can start applying its own runtime control logic by interacting with One-Stop-API GUI.

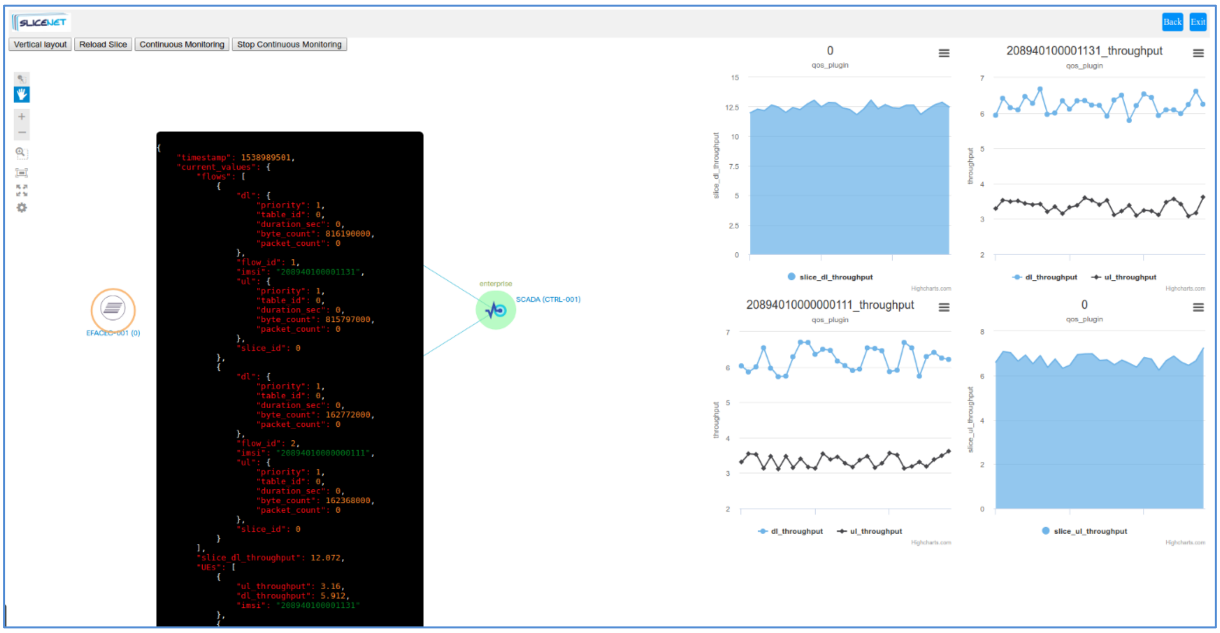


```
{
    "imsi": "208940100000000111",
    "ip": "172.16.0.6",
    "location": "enb-234881024",
    "status": "ATTACHED"
}
```

Figure 39 New UE (IED) addition to the slice and status information retrieval

Figure 39 shows the execution of the slice expansion operation by (emulating, since there is only one real IED in the testbed) the addition of a new IED/UE to the network slice, given its IMSI. As a result, a new IED icon appears, as a proof that the vertical tailored slice view has been dynamically updated following the workflow in section 4.2.2. Figure 39 also shows the result of the subsequent retrieval of IED/UE status information (in the black box prompted by the GUI) that indicates that new (emulated) IED/UE is successfully attached to the slice instance.

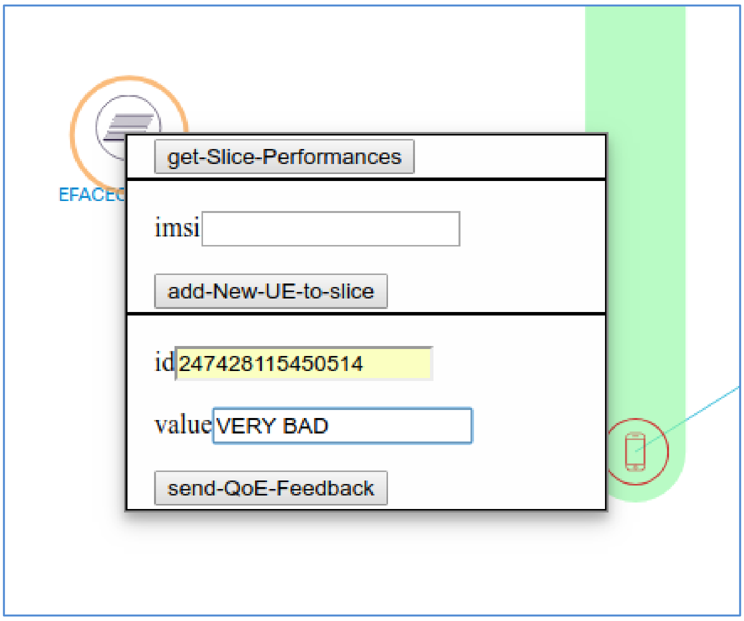
Figure 40 shows the monitoring capability provided by the QoS monitoring control function, both enforcing an explicit monitoring query (in the black box) and a continuous monitoring via websocket that allows the One-Stop-API GUI to plot the slice throughput



```
{
    "timestamp": 1538989501,
    "current_values": {
        "flows": [
            {
                "dl": {
                    "priority": 1,
                    "table_id": 0,
                    "duration_sec": 0,
                    "byte_count": 816190000,
                    "packet_count": 0
                },
                "flow_id": 1,
                "imsi": "208940100001131",
                "ul": {
                    "priority": 1,
                    "table_id": 0,
                    "duration_sec": 0,
                    "byte_count": 815797000,
                    "packet_count": 0
                },
                "slice_id": 0
            },
            {
                "dl": {
                    "priority": 1,
                    "table_id": 0,
                    "duration_sec": 0,
                    "byte_count": 162772000,
                    "packet_count": 0
                },
                "flow_id": 2,
                "imsi": "208940100000000111",
                "ul": {
                    "priority": 1,
                    "table_id": 0,
                    "duration_sec": 0,
                    "byte_count": 162368000,
                    "packet_count": 0
                },
                "slice_id": 0
            }
        ],
        "slice_dl_throughput": 12.072,
        "UEs": [
            {
                "ul_throughput": 3.16,
                "dl_throughput": 5.912,
                "imsi": "208940100001131"
            },
            {
```

Figure 40 Slice and IEDs throughput monitoring

Finally, as shown in Figure 41, the P&P instance allows the vertical to express a QoE feedback (at slice level) that is then processed by the P&P QoE optimizer control function to trigger a slice ptimization process.



imsi

id 247428115450514

value VERY BAD

Figure 41 QoE feedback enforcement through One-Stop-API GUI

6 Conclusions

This deliverable has presented the design and prototype implementation of the SliceNet Plug & Play control framework, which allows dedicated runtime slice control instances to be offered to verticals by providing the required slice views, control endpoints and APIs. The main goal of the Plug & Play control framework is to advance the current practices implemented by service providers and network operators for their service and slice offerings, which are currently mostly based on very limited runtime control and management exposure of slices and services to their customers. Moreover, while other 5GPPP Phase 2 projects are tackling with the involvement of vertical actors in the design phase of end-to-end network slices and services, capturing at different levels their requirements and often offering DevOps tools to ease the preparation of new specialized slice and service templates and descriptors, SliceNet aims at involving verticals and slice consumers also in the runtime phase of slices lifecycle as a further means of vertical-tailored customization.

The Plug & Play control addresses these SliceNet challenges by providing a common framework for offering to verticals a runtime view of their slices that fulfil their heterogeneous business logics and requirements while hiding those service and infrastructure capabilities that slice providers do not want to expose. For this, a very flexible, scalable and dynamic design approach has been chosen, where Plug & Play control instances are a collection of microservices, with each microservice providing a very specific set of functions and logic for creating a highly customized control environment to be offered for each slice to verticals and slice consumers. These microservices belong to three main categories: i) vertical tailored REST API microservices, actually exposing the vertical-tailored customized slice view and REST APIs to the vertical, ii) the core abstraction microservices maintaining the binding between the northbound runtime API and the capabilities offered by plugins and control functions, practically implementing the customized slice control exposure, and iii) control function and plugin microservices, which implement specific vertical-tailored control logics on top of an abstraction of SliceNet platform data stores, control and management APIs. The common ground of each Plug & Play control instance is the technology-agnostic slice exposure information model, that abstracts the SliceNet slice information model and enable the description of any vertical service, slice, network functions and resource elements in a common way through a generalized topology graph approach. On top of this model, the Plug & Play instances offer northbound APIs to verticals as the space for customized runtime slice control. Here, the interaction with verticals is mediated by the One-Stop-API framework, which enables authorization, authentication and role based access features on top of the Plug & Play control instances.

The SliceNet Plug & Play control software prototype has been also documented in this deliverable, in terms of reference open source frameworks, implementation choices and preliminary integration and validation. In summary, the Plug & Play control has been implemented from scratch as a containerized application, where microservices composing a given instance are deployed as Docker containers that are orchestrated by Kubernetes. In particular, the combined use of Docker Engine and Kubernetes leverages on two de-facto standard open source tools for the deployment and orchestration of microservice based applications, enabling flexibility, scalability and in the Plug & Play deployments. A preliminary integration and validation of the Plug & Play control framework has been performed in the context of the SmartGrid use case Phase 1 in the Altice Labs testbed in Aveiro.

As part of the next steps, the Plug & Play control framework will be integrated within the SliceNet platform in the context of WP8 activities. While the integration with the One-Stop-API framework is already in a mature phase, the interactions with SliceNet orchestration and control plane functionalities is one of main targets for the short term, therefore requiring a coordination with ongoing activities in WP4 and WP7. Moreover, the Plug & Play requirements from SmartGrid, SmartCity and eHealth use cases will be further investigated with the use case teams to devise additional plugins and control functions required to fulfil the SliceNet verticals' needs. Finally, the lifecycle management of the Plug & Play control instances, currently performed manually with

Kubernetes command line tools, will be fully automated through the Plug & Play manager, which will also be responsible to align coordinate Plug & Play workflows with slice and service orchestration logics.

References

[1] SliceNet Deliverable D2.2, “Overall Architecture and Interfaces Definition”, SliceNet Consortium, January 2018

[2] SliceNet Deliverable D2.3, “Control Plane System Definition, APIs and Interfaces”, SliceNet Consortium, April 2018

[3] 5GPPP Phase 2 – 5G Transformer, http://5g-transformer.eu/

[4] 5GPPP Phase 2 – MATILDA, http://www.matilda-5g.eu/

[5] 5GPPP Phase 2 – 5GCity, https://www.5gcity.eu

[6] 5GCity Blog Post, https://www.5gcity.eu/wp-content/uploads/2018/04/Nxw-Blog-5GCity.pdf

[7] 5GPPP Phase 2 – 5G ESSENCE, www.5g-essence-h2020.eu

[8] 5GPPP Phase 1 – 5G SESAME, http://www.sesame-h2020-5g-ppp.eu

[9] 5G SESAME D3.4, “CESC Small Cell prototype and PoC”,http://www.sesame-h2020-5g-ppp.eu/Portals/0/Deliverables/Deliverable%203.4_v1.0_final.pdf?ver=2018-05-18-143714-370

[10] 5GPPP Phase 2 – 5G MEDIA, www.5gmedia.eu

[11] 5G MEDIA D2.3, “5G-MEDIA Platform Architecture”, http://www.5gmedia.eu/cms/wp-content/uploads/2018/08/5G-MEDIA-D2.3-5G-MEDIA-Platform-Architecture_v1.0_final.pdf

[12] ONF, “SDN architecture”, ONF TR-502, June 2014.

[13] ONF, “OpenFlow Switch Specification”, https://www.opennetworking.org/software-defined-standards/specifications/

[14] R. Enns, Ed., et .al, “Network Configuration Protocol (NETCONF)”, IETF RFC 6241, June 2011

[15] ETSI GS NFV-MAN 001 V1.1.1, “Network Functions Virtualisation (NFV); Management and Orchestration”, ETSI NFV ISG, December 2014

[16] SliceNet Deliverable D2.4, “Management Plane System Definition, APIs and Interfaces”, SliceNet Consortium, May 2018

[17] Docker, https://www.docker.com/

[18] Kubernetes, www.kubernetes.io

[19] OpenAPI initiative, https://www.openapis.org/

[20] SliceNet Deliverable D3.4, “Design and Prototyping of Integrated Multi-domain SliceNet Architecture”, SliceNet Consortium, July 2018

[21] SliceNet Deliverable D2.1, “Vertical Sector Requirements Analysis and Use Case Definition”, SliceNet Consortium, November 2017

[22] I. Fette, A. Melnikov, “The WebSocket Protocol”, RFC 6455, December 2011, https://tools.ietf.org/html/rfc6455

[23] SliceNet Deliverable D6.3, “Management for the Plug & Play Control Plane”, SliceNet consortium, planned for May 2019.

[24] SliceNet Deliverable D7.1, “Cross-Plane Slice and Service Orchestrator”, SliceNet consortium, planned for September 2019

[25] REST HATEOAS approach, https://en.wikipedia.org/wiki/HATEOAS

[26] SliceNet Deliverable D4.3, “Network Slicing in Multi-tenant Virtualised Single-Domain Infrastructures”, SliceNet consortium, planned for November 2018.

[27] 3GPP TS 28.530; Telecommunication management; Management of 5G networks and network slicing; Concepts, use cases and requirements, v15.0.0, September 2018

[28] Nextworks Slicer, https://github.com/nextworks-it/slicer

[29] Nextworks TIMEO, https://github.com/nextworks-it/timeo

[30] SliceNet Deliverable D8.1, “Northbound API Specification and Graphical Interface (Iteration I)”, SliceNet consortium, planned for May 2019

[31] Tornado Web Server, https://www.tornadoweb.org/en/stable/

[32] Python Requests, http://docs.python-requests.org/en/master/#

[33] Python Swagger-parser, https://pypi.org/project/swagger-parser/

[34] Java Spring, https://spring.io/

[35] Apache Maven, https://maven.apache.org/

[36] OpenAirInterface RAN, https://gitlab.eurecom.fr/oai/openairinterface5g/

[37] OpenAirInterface Core Network, https://github.com/OPENAIRINTERFACE/openair-cn

[38] Mosaic5G FlexRAN, http://mosaic-5g.io/apidocs/flexran/

[39] Mosaic5G LL-MEC, http://mosaic-5g.io/apidocs/ll-mec/

Annex A.1 Plug & Play internal information model JSON schema

```
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "$id": "http://json-schema.org/draft-07/schema#",
  "title": "Plug & Play Slice internal",
  "type": "object",
  "properties": {
    "domain_type": {
      "$id": "/properties/domain_type",
      "type": "string",
      "title": "single/multi-domain slice",
      "default": "",
      "examples": [
        "single"
      ],
      "enum": [
        "single",
        "multi"
      ]
    },
    "elements": {
      "$id": "/properties/elements",
      "type": "array",
      "title": "slice elements",
      "minItems": 1,
      "items": {
        "$id": "/properties/elements/items",
        "type": "object",
        "properties": {
          "properties": {
            "$id": "/properties/elements/items/properties/properties",
            "type": "array",
            "title": "slice element properties",
            "items": {
              "$id": "/properties/elements/items/properties/properties/items",
              "type": "object",
              "properties": {
                "port": {
                  "$id": "/properties/elements/items/properties/properties/items/properties/port",
                  "type": "string",
                  "title": "service port",
                  "default": ""
                },
                "allowed_api": {
                  "$id":
"/properties/elements/items/properties/properties/items/properties/allowed_api",
                  "type": "array",
                  "title": "exposed apis",
                  "minItems": 1,
                  "items": {
                    "$id":
"/properties/elements/items/properties/properties/items/properties/allowed_api/items",
                    "type": "object",
                    "properties": {
                      "forwarding": {
                        "$id":
"/properties/elements/items/properties/properties/items/properties/allowed_api/items/properties/forward
ing",
                        "type": "string",
                        "title": "api mapping type",
                        "description": "The mapping type applied to P&P drivers APIs",
                        "default": "",
                        "enum": [
                          "direct",
                          "wired"
                        ]
                      },
                      "api_id": {
                        "$id":
"/properties/elements/items/properties/properties/items/properties/allowed_api/items/properties/api_id"
,
                        "type": "string",
                        "title": "api operation id",
                        "description": "The reference to the P&P driver openAPI operation",
                        "default": ""
                      }
                    },
                    "required": [
                      "forwarding",
                      "api_id"
                    ]
                  }
                },
                "category": {
                  "$id": "/properties/elements/items/properties/properties/items/properties/category",
                  "type": "string",
                  "title": "apis class/family",
                  "default": "",
                  "enum": [
                    "control",
                    "management",
                    "monitoring"
                  ]
                },
                "type": {
                  "$id": "/properties/elements/items/properties/properties/items/properties/type",
                  "type": "string",
                  "title": "service type",
                  "default": "",
                  "enum": [
                    "plugin",
                    "other"
                  ]
                },
                "property_id": {
                  "$id":
"/properties/elements/items/properties/properties/items/properties/property_id",
                  "type": "string",
                  "title": "service/plugin id",
                  "description": "Reference to the P&P plugin service name",
                  "default": ""
                }
              },
              "required": [
                "allowed_api",
                "category",
                "type",
                "property_id"
              ]
            }
          },
          "element_id": {
            "$id": "/properties/elements/items/properties/element_id",
            "type": "string",
            "title": "slice element id",
            "description": "The Unique ID of the slice element",
            "default": "",
          },
          "context": {
            "$id": "/properties/elements/items/properties/context",
            "type": "string",
            "title": "slice element family/class",
            "default": "",
            "enum": [
              "RAN",
              "SLICE",
              "UE",
              "APP",
              "CORE",
              "VNF",
              "PNF",
              "NS",
              "MECApp",
              "VL",
              "INTERDOMAIN",
              "TRANSPORT"
            ]
          },
          "parent": {
            "$id": "/properties/elements/items/properties/parent",
            "type": "string",
            "title": "slice element parent id",
            "description": "The parent element in this slice view",
            "default": ""
          },
          "domain_id": {
            "$id": "/properties/elements/items/properties/domain_id",
            "type": "string",
            "title": "slice element domain",
            "default": ""
          },
          "level": {
            "$id": "/properties/elements/items/properties/level",
            "type": "string",
            "title": "slice element abstraction",
            "default": "",
            "enum": [
              "Service",
              "Slice",
              "Subslice",
              "NF"
            ]
          },
          "location": {
            "$id": "/properties/elements/items/properties/location",
            "type": "string",
            "title": "slice element location",
            "default": ""
          },
          "type": {
            "$id": "/properties/elements/items/properties/type",
            "type": "string",
            "title": "slice element type",
            "default": "",
            "enum": [
              "node",
              "link"
            ]
          }
        },
        "required": [
          "element_id",
          "context",
          "domain_id",
          "location",
          "type"
        ]
      }
    },
    "slice_id": {
      "$id": "/properties/slice_id",
      "type": "string",
      "title": "slice id",
      "description": "The unique ID of the slice",
      "default": ""
    },
    "owner_id": {
      "$id": "/properties/owner_id",
      "type": "string",
      "title": "slice owner id",
      "description": "The unique ID of the slice owner",
      "default": ""
    },
    "topology": {
      "$id": "/properties/topology",
      "type": "array",
      "title": "slice topology",
      "minItems": 1,
      "items": {
        "$id": "/properties/topology/items",
        "type": "object",
        "properties": {
          "adjacent_elements": {
            "$id": "/properties/topology/items/properties/adjacent_elements",
            "type": "array",
            "items": {
              "$id": "/properties/topology/items/properties/adjacent_elements/items",
              "type": "object",
              "properties": {
                "element_id": {
                  "$id":
"/properties/topology/items/properties/adjacent_elements/items/properties/element_id",
                  "type": "string",
                  "title": "node id",
                  "default": ""
                },
                "link_id": {
                  "$id":
"/properties/topology/items/properties/adjacent_elements/items/properties/link_id",
                  "type": "string",
                  "title": "link id",
                  "default": ""
                }
              }
            }
          },
          "element_id": {
            "$id": "/properties/topology/items/properties/element_id",
            "type": "string",
            "title": "node id",
            "default": ""
          }
        },
        "required": [
          "element_id"
        ]
      }
    }
  },
  "required": [
    "domain_type",
    "elements",
    "slice_id",
    "owner_id",
    "topology"
  ]
}
```

Snippet Plug & Play slice internal JSON schema

Annex A.2 Example of Plug & Play internal slice control view

```
{
    "slice_id": "0",
    "domain_type": "single",
    "owner_id": "efacec",
    "elements": [
        {
            "type": "node",
            "element_name": "EFACEC-001",
            "context": "SLICE",
            "element_id": "0",
            "level": "Slice",
            "domain_id": "alb",
            "location":"portugal",
            "properties": [
                {
                    "type": "plugin",
                    "property_id": "qos_plugin",
                    "category": "monitoring",
                    "port": "65007",
                    "allowed_api": [
                        {
                            "api_id": "get-Slice-Performances",
                            "forwarding": "direct"
                        }
                    ]
                },
                {
                    "type": "plugin",
                    "property_id": "plug_and_play_ue_ctrl_plugin",
                    "category": "management",
                    "port": "65006",
                    "allowed_api": [
                        {
                            "api_id": "add-New-UE-to-slice",
                            "forwarding": "direct",
                        }
                    ]
                },
                {
                    "type": "plugin",
                    "property_id": "qoe_plugin_v1",
                    "category": "management",
                    "port": "65005",
                    "allowed_api": [
                        {
                            "api_id": "send-QoE-Feedback",
                            "forwarding": "direct",
                        }
                    ]
                }
            ]
        },
        {
            "type": "node",
            "element_name": "IED-1",
            "location": "edge_aveiro",
            "context": "UE",
            "element_id": "208940100001131",
            "domain_id": "domain-001",
            "properties": [
                {
                    "type": "plugin",
                    "property_id": "plug_and_play_ue_ctrl_plugin",
                    "category": "monitoring",
                    "port": "65006",
                    "allowed_api": [
                        {
                            "api_id": "get-UE-info",
                            "forwarding": "direct",
                        }
                    ]
                }
            ]
        },
        {
            "type": "node",
            "element_name": "SCADA",
            "location": "enterprise",
            "context": "APP",
            "element_id": "CTRL-001",
            "level": "Service",
            "domain_id": "domain-001",
            "properties": []
        },
        {
            "type": "link",
            "element_name": "LINK-1",
            "location": "edge_aveiro",
            "context": "VL",
            "element_id": "LINK-1",
            "level": "Service",
            "domain_id": "domain-001",
            "properties": []
        },
        {
            "type": "node",
            "element_name": "IED-3",
            "location": "edge_aveiro",
            "context": "UE",
            "element_id": "1234567890",
            "domain_id": "domain-001",
            "properties": [
                {
                    "type": "plugin",
                    "property_id": "plug_and_play_ue_ctrl_plugin",
                    "category": "monitoring",
                    "port": "65006",
                    "allowed_api": [
                        {
                            "api_id": "get-UE-info",
                            "forwarding": "direct",
                        }
                    ]
                }
            ]
        },
        {
            "type": "link",
            "element_name": "LINK-3",
            "location": "edge_aveiro",
            "context": "VL",
            "element_id": "LINK-3",
            "level": "Service",
            "domain_id": "domain-001",
            "properties": []
        }
    ],
    "topology": [
        {
            "adjacent_elements": [
                {
                    "element_id": "208940100001131",
                    "link_id": "LINK-1"
                },
                {
                    "element_id": "1234567890",
                    "element_id": "LINK-3"
                }
            ],
            "element_id": "SCADA"
        }
    ]
}
```

Annex B.1 Plug & Play northbound information model JSON schema

```
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "$id": "http://json-schema.org/draft-07/schema#",
  "title": "Plug & Play Slice internal",
  "type": "object",
  "properties": {
    "domain_type": {
      "$id": "/properties/domain_type",
      "type": "string",
      "title": "single/multi-domain slice",
      "default": "",
      "examples": [
        "single"
      ],
      "enum": [
        "single",
        "multi"
      ]
    },
    "elements": {
      "$id": "/properties/elements",
      "type": "array",
      "title": "slice elements",
      "minItems": 1,
      "items": {
        "$id": "/properties/elements/items",
        "type": "object",
        "properties": {
          "properties": {
            "$id": "/properties/elements/items/properties/properties",
            "type": "array",
            "title": "slice element properties",
            "items": {
              "$id": "/properties/elements/items/properties/properties/items",
              "type": "object",
              "properties": {
                "port": {
                  "$id": "/properties/elements/items/properties/properties/items/properties/port",
                  "type": "string",
                  "title": "service port",
                  "default": "",
                  "examples": [
                    "65002"
                  ]
                },
                "allowed_api": {
                  "$id":
"/properties/elements/items/properties/properties/items/properties/allowed_api",
                  "type": "array",
                  "title": "exposed apis",
                  "minItems": 1,
                  "items": {
                    "$id":
"/properties/elements/items/properties/properties/items/properties/allowed_api/items",
                    "type": "object",
                    "properties": {
                      "forwarding": {
                        "$id":
"/properties/elements/items/properties/properties/items/properties/allowed_api/items/properties/forward
ing",
                        "type": "string",
                        "title": "api mapping type",
                        "description": "The mapping type applied to P&P drivers APIs",
                        "default": "",
                        "examples": [
                          "direct"
                        ],
                        "enum": [
                          "direct",
                          "wired"
                        ]
                      },
                      "api_id": {
                        "$id":
"/properties/elements/items/properties/properties/items/properties/allowed_api/items/properties/api_id"
,
                        "type": "string",
                        "title": "api operation id",
                        "description": "The reference to the P&P driver openAPI operation",
                        "default": "",
                        "examples": [
                          "getConf"
                        ]
                      },
                      "parameters": {
                        "$id":
"/properties/elements/items/properties/properties/items/properties/allowed_api/items/properties/paramet
ers",
                        "type": "array",
                        "title": "api parameters",
                        "items": {
                          "$id":
"/properties/elements/items/properties/properties/items/properties/allowed_api/items/properties/paramet
ers/items",
                          "type": "object",
                          "properties": {
                            "description": {
                              "$id":
"/properties/elements/items/properties/properties/items/properties/allowed_api/items/properties/paramet
ers/items/properties/description",
                              "type": "string",
                              "title": "parameter description",
                              "default": "",
                              "examples": [
                                "Name of the node"
                              ]
                            },
                            "in": {
                              "$id":
"/properties/elements/items/properties/properties/items/properties/allowed_api/items/properties/paramet
ers/items/properties/in",
                              "type": "string",
                              "title": "position",
                              "default": "",
                              "examples": [
                                "path"
                              ],
                              "enum": [
                                "path",
                                "body"
                              ]
                            },
                            "name": {
                              "$id":
"/properties/elements/items/properties/properties/items/properties/allowed_api/items/properties/paramet
ers/items/properties/name",
                              "type": "string",
                              "title": "parameter name",
                              "default": "",
                              "examples": [
                                "nodeName"
                              ]
                            },
                            "schema": {
                              "$id":
"/properties/elements/items/properties/properties/items/properties/allowed_api/items/properties/paramet
ers/items/properties/schema",
                              "type": "object",
                              "title": "body schema",
                              "properties": {
                                "type": {
                                  "$id":
"/properties/elements/items/properties/properties/items/properties/allowed_api/items/properties/paramet
ers/items/properties/schema/properties/type",
                                  "type": "string",
                                  "title": "schema type",
                                  "default": "",
                                  "examples": [
                                    "object"
                                  ]
                                }
                              }
                            },
                            "type": {
                              "$id":
"/properties/elements/items/properties/properties/items/properties/allowed_api/items/properties/paramet
ers/items/properties/type",
                              "type": "string",
                              "title": "parameter type",
                              "default": "",
                              "examples": [
                                "string"
                              ]
                            },
                            "required": {
                              "$id":
"/properties/elements/items/properties/properties/items/properties/allowed_api/items/properties/paramet
ers/items/properties/required",
                              "type": "boolean",
                              "title": "required",
                              "default": false,
                              "examples": [
                                true
                              ]
                            }
                          }
                        }
                      },
                      "operation_type": {
                        "$id":
"/properties/elements/items/properties/properties/items/properties/allowed_api/items/properties/operati
on_type",
                        "type": "string",
                        "title": "api operation type",
                        "default": "",
                        "examples": [
                          "get"
                        ],
                        "enum": [
                          "post",
                          "get",
                          "put",
                          "patch",
                          "delete"
                        ]
                      },
                      "endpoint": {
                        "$id":
"/properties/elements/items/properties/properties/items/properties/allowed_api/items/properties/endpoin
t",
                        "type": "string",
                        "title": "api endpoint",
                        "default": "",
                        "examples": [
                          "/plug-and-play-test/management/NF-
001/test_plugin_2/getConf/?nodeName={nodeName}"
                        ]
                      }
                    },
                    "required": [
                      "forwarding",
                      "api_id"
                    ]
                  }
                },
                "category": {
                  "$id": "/properties/elements/items/properties/properties/items/properties/category",
                  "type": "string",
                  "title": "apis class/family",
                  "default": "",
                  "examples": [
                    "management"
                  ],
                  "enum": [
                    "control",
                    "management",
                    "monitoring"
                  ]
                },
                "type": {
                  "$id": "/properties/elements/items/properties/properties/items/properties/type",
                  "type": "string",
                  "title": "service type",
                  "default": "",
                  "examples": [
                    "plugin"
                  ],
                  "enum": [
                    "plugin",
                    "other"
                  ]
                },
                "property_id": {
                  "$id":
"/properties/elements/items/properties/properties/items/properties/property_id",
                  "type": "string",
                  "title": "service/plugin id",
                  "description": "Reference to the P&P plugin service name",
                  "default": "",
                  "examples": [
                    "test_plugin_2"
                  ]
                }
              },
              "required": [
                "allowed_api",
                "category",
                "type",
                "property_id"
              ]
            }
          },
          "element_id": {
            "$id": "/properties/elements/items/properties/element_id",
            "type": "string",
            "title": "slice element id",
            "description": "The Unique ID of the slice element",
            "default": "",
            "examples": [
              "NF-001"
            ]
          },
          "context": {
            "$id": "/properties/elements/items/properties/context",
            "type": "string",
            "title": "slice element family/class",
            "default": "",
            "examples": [
              "VNF"
            ],
            "enum": [
              "RAN",
              "SLICE",
              "UE",
              "APP",
              "CORE",
              "VNF",
              "PNF",
              "NS",
              "MECApp",
              "VL",
              "INTERDOMAIN",
              "TRANSPORT"
            ]
          },
          "parent": {
            "$id": "/properties/elements/items/properties/parent",
            "type": "string",
            "title": "slice element parent id",
            "description": "The parent element in this slice view",
            "default": "",
            "examples": [
              "EDGE-001"
            ]
          },
          "domain_id": {
            "$id": "/properties/elements/items/properties/domain_id",
            "type": "string",
            "title": "slice element domain",
            "default": "",
            "examples": [
              "domain-001"
            ]
          },
          "level": {
            "$id": "/properties/elements/items/properties/level",
            "type": "string",
            "title": "slice element abstraction",
            "default": "",
            "examples": [
              "Service"
            ],
            "enum": [
              "Service",
              "Slice",
              "Subslice",
              "NF"
            ]
          },
          "location": {
            "$id": "/properties/elements/items/properties/location",
            "type": "string",
            "title": "slice element location",
            "default": "",
            "examples": [
              "edge_1"
            ]
          },
          "type": {
            "$id": "/properties/elements/items/properties/type",
            "type": "string",
            "title": "slice element type",
            "default": "",
            "examples": [
              "node"
            ],
            "enum": [
              "node",
              "link"
            ]
          }
        },
        "required": [
          "element_id",
          "context",
          "domain_id",
          "location",
          "type"
        ]
      }
    },
    "slice_id": {
      "$id": "/properties/slice_id",
      "type": "string",
      "title": "slice id",
      "description": "The unique ID of the slice",
      "default": "",
      "examples": [
        "slice-002"
      ]
    },
    "owner_id": {
      "$id": "/properties/owner_id",
      "type": "string",
      "title": "slice owner id",
      "description": "The unique ID of the slice owner",
      "default": "",
      "examples": [
        "tenant-002"
      ]
    },
    "topology": {
      "$id": "/properties/topology",
      "type": "array",
      "title": "slice graph",
      "minItems": 1,
      "items": {
        "$id": "/properties/topology/items",
        "type": "object",
        "properties": {
          "adjacent_elements": {
            "$id": "/properties/topology/items/properties/adjacent_elements",
            "type": "array",
            "items": {
              "$id": "/properties/topology/items/properties/adjacent_elements/items",
              "type": "object",
              "properties": {
                "element_id": {
                  "$id": 
"/properties/topology/items/properties/adjacent_elements/items/properties/element_id",
                  "type": "string",
                  "title": "node id",
                  "default": "",
                  "examples": [
                    "NF-002"
                  ]
                },
                "link_id": {
                  "$id": 
"/properties/topology/items/properties/adjacent_elements/items/properties/link_id",
                  "type": "string",
                  "title": "link id",
                  "default": "",
                  "examples": [
                    "LINK-001"
                  ]
                }
              }
            }
          },
          "element_id": {
            "$id": "/properties/topology/items/properties/element_id",
            "type": "string",
            "title": "node id",
            "default": "",
            "examples": [
              "NF-001"
            ]
          }
        },
        "required": [
          "element_id"
        ]
      }
    }
  },
  "definitions": {
    "$id": "/properties/definitions",
    "type": "object",
    "title": "definitions",
     "default": "",
     "properties": {
       "<schema_name>": {
         "$id": "/properties/definitions/<schema_name>",
         "type": "object",
         "required": [
           "type",
           "properties"
         ],

        "properties": {
          "type": {
            "$id": "#/properties/definitions/properties/<schema_name>/properties/type",
            "type": "string",
            "title": "type",
            "default": "",
            "examples": [
              "object"
            ]
          },
          "properties": {
            "$id": "#/properties/definitions/properties/<schema_name>/properties/properties",
            "type": "object",
            "title": "properties",
          }
        }
      }
    }
  },
  "required": [
    "domain_type",
    "elements",
    "slice_id",
    "owner_id",
    "topology"
  ]
}
```

Snippet 10 Plug & Play slice NBI JSON schema

Annex B.2 Example of Plug & Play northbound slice control view

```
{
  "slice_id":"0",
  "domain_type":"single",
  "owner_id":"efacec",
  "elements":[
    {
      "type":"node",
      "element_name":"EFACEC-001",
      "context":"SLICE",
      "element_id":"0",
      "level":"Slice",
      "domain_id": "alb",
      "location": "portugal",
      "properties":[
        {
          "type":"plugin",
          "property_id":"qos_plugin",
          "category":"monitoring",
          "port":"65007",
          "allowed_api":[
            {
              "operation_type":"get",
              "api_id":"get-Slice-Performances",
              "forwarding":"direct",
              "parameters":[
                {}
              ],
              "responses":{
                "200":{
                  "description":"OK",
                  "schema":{
                    "$ref":"#/definitions/8b503a43b0245d423d4bf5b52ae3ea03"
                  }
                },
                "404":{
                  "description":"Not Found"
                }
              },
              "endpoint":"/plug-and-play-test/monitoring/0/qos_plugin/get-Slice-Performances/"
            }
          ]
        },
        {
          "type":"plugin",
          "property_id":"plug_and_play_ue_ctrl_plugin",
          "category":"management",
          "port":"65006",
          "allowed_api":[
            {
              "operation_type":"post",
              "api_id":"add-New-UE-to-slice",
              "forwarding":"direct",
              "parameters":[
                {
                  "request":{
                    "name":"request",
                    "description":"request",
                    "schema":{
                      "$ref":"#/definitions/7d25f488f9328fc5dc72d13872944daf"
                    },
                    "in":"body",
                    "required":true
                  }
                }
              ],
              "responses":{
                "200":{
                  "description":"OK",
                  "schema":{
                    "type":"string"
                  }
                },
                "201":{
                  "description":"Created",
                  "schema":{
                    "type":"string"
                  }
                },
                "400":{
                  "description":"Bad Request"
                },
                "401":{
                  "description":"Unauthorized"
                },
                "403":{
                  "description":"Forbidden"
                },
                "404":{
                  "description":"Not Found"
                }
              },
              "endpoint":"/plug-and-play-test/management/0/plug_and_play_ue_ctrl_plugin/add-New-UE-to-
slice/"
            }
          ]
        },
        {
          "type":"plugin",
          "property_id":"qoe_plugin_v1",
          "category":"management",
          "port":"65005",
          "allowed_api":[
            {
              "operation_type":"put",
              "api_id":"send-QoE-Feedback",
              "forwarding":"direct",
              "parameters":[
                {
                  "feedback":{
                    "name":"feedback",
                    "description":"feedback",
                    "schema":{
                      "$ref":"#/definitions/d7590678b7a09c9aa2cb27617dd0b3be"
                    },
                    "in":"body",
                    "required":true
                  }
                }
              ],
              "responses":{
                "200":{
                  "description":"OK"
                },
                "400":{
                  "description":"Bad Request"
                }
              },
              "endpoint":"/plug-and-play-test/management/0/qoe_plugin_v1/send-QoE-Feedback/"
            }
          ]
        }
      ]
    },
    {
      "type":"node",
      "element_name":"IED-1",
      "location":"edge_aveiro",
      "context":"UE",
      "element_id":"208940100001131",
      "domain_id":"domain-001",
      "properties":[
        {
          "type":"plugin",
          "property_id":"plug_and_play_ue_ctrl_plugin",
          "category":"monitoring",
          "port":"65006",
          "allowed_api":[
            {
              "operation_type":"get",
              "api_id":"get-UE-info",
              "forwarding":"direct",
              "parameters":[
                {
                  "ueId":{
                    "name":"ueId",
                    "description":"ueId",
                    "type":"string",
                    "in":"path",
                    "required":true
                  }
                }
              ],
              "responses":{
                "200":{
                  "description":"Success",
                  "schema":{
                    "$ref":"#/definitions/bb85cbf2a781c923731efc3423dd7528",
                    "x-scope":[
                      ""
                    ]
                  }
                },
                "401":{
                  "description":"Unauthorized"
                },
                "403":{
                  "description":"Forbidden"
                },
                "404":{
                  "description":"Not Found"
                }
              },
              "endpoint":"/plug-and-play-
test/monitoring/208940100001131/plug_and_play_ue_ctrl_plugin/get-UE-info/?ueId={ueId}"
            }
          ]
        }
      ]
    },
    {
      "type":"node",
      "element_name":"SCADA",
      "location":"enterprise",
      "context":"APP",
      "element_id":"CTRL-001",
      "level":"Service",
      "domain_id":"domain-001",
      "properties":[

      ]
    },
    {
      "type":"link",
      "element_name":"LINK-1",
      "location":"edge_aveiro",
      "context":"VL",
      "element_id":"LINK-1",
      "level":"Service",
      "domain_id":"domain-001",
      "properties":[

      ]
    },
    {
      "type":"node",
      "element_name":"IED-3",
      "location":"edge_aveiro",
      "context":"UE",
      "element_id":"1234567890",
      "domain_id":"domain-001",
      "properties":[
        {
          "type":"plugin",
          "property_id":"plug_and_play_ue_ctrl_plugin",
          "category":"control",
          "port":"65006",
          "allowed_api":[
            {
              "operation_type":"get",
              "api_id":"get-UE-info",
              "forwarding":"direct",
              "parameters":[
                {
                  "ueId":{
                    "name":"ueId",
                    "description":"ueId",
                    "type":"string",
                    "in":"path",
                    "required":true
                  }
                }
              ],
              "responses":{
                "200":{
                  "description":"Success",
                  "schema":{
                    "$ref":"#/definitions/bb85cbf2a781c923731efc3423dd7528",
                    "x-scope":[
                      ""
                    ]
                  }
                },
                "401":{
                  "description":"Unauthorized"
                },
                "403":{
                  "description":"Forbidden"
                },
                "404":{
                  "description":"Not Found"
                }
              },
              "endpoint":"/plug-and-play-test/monitoring/1234567890/plug_and_play_ue_ctrl_plugin/get-
UE-info/?ueId={ueId}"
            }
          ]
        }
      ]
    },
    {
      "type":"link",
      "element_name":"LINK-3",
      "location":"edge_aveiro",
      "context":"VL",
      "element_id":"LINK-3",
      "level":"Service",
      "domain_id":"domain-001",
      "properties":[

      ]
    }
  ],
  "topology":[
    {
      "adjacent_elements":[
        {
          "element_id":" 208940100001131 ",
          "link_id":"LINK-1"
        },
        {
          "element_id":"1234567890",
          "link_id":"LINK-3"
        }
      ],
      "element_id":"SCADA"
    }
  ],
  "definitions":{
    "7d25f488f9328fc5dc72d13872944daf":{
      "type":"object",
      "properties":{
        "imsi":{
          "type":"string"
        }
      }
    },
    "d7590678b7a09c9aa2cb27617dd0b3be":{
      "type":"object",
      "properties":{
        "id":{
          "type":"string"
        },
        "value":{
          "type":"string"
        }
      }
    },
    "bb85cbf2a781c923731efc3423dd7528":{
      "type":"object",
      "properties":{
        "imsi":{
          "type":"string"
        },
        "ip":{
          "type":"string"
        },
        "location":{
          "type":"string"
        },
        "status":{
          "type":"string"
        }
      }
    },
    "8b503a43b0245d423d4bf5b52ae3ea03":{
      "type":"object",
      "properties":{
        "average_values":{
          "type":"object",
          "items":{
            "$ref":"#/definitions/8b503a43b0245d423d4bf5b52ae3ea03_1"
          }
        },
        "timestamp":{
          "type":"integer"
        },
        "slice_id":{
          "type":"string"
        },
        "current_values":{
          "type":"object",
          "properties":{
            "flows":{
              "type":"array",
              "items":{
                "$ref":"#/definitions/8b503a43b0245d423d4bf5b52ae3ea03_2"
              }
            },
            "UEs":{
              "type":"array",
              "items":{
                "$ref":"#/definitions/8b503a43b0245d423d4bf5b52ae3ea03_3"
              }
            },
            "slice_dl_throughput":{
              "type":"number"
            },
            "slice_ul_throughput":{
              "type":"number"
            }
          }
        }
      }
    },
    "8b503a43b0245d423d4bf5b52ae3ea03_1":{
      "properties":{
        "UEs":{
          "type":"array",
          "items":{
            "$ref":"#/definitions/8b503a43b0245d423d4bf5b52ae3ea03_4"
          }
        },
        "slice_avg_dl_throughput":{
          "type":"number"
        },
        "slice_avg_ul_throughput":{
          "type":"number"
        }
      }
    },
    "8b503a43b0245d423d4bf5b52ae3ea03_2":{
      "properties":{
        "flow_id":{
          "type":"integer"
        },
        "imsi":{
          "type":"string"
        },
        "slice_id":{
          "type":"integer"
        },
        "dl":{
          "type":"object",
          "items":{
            "$ref":"#/definitions/8b503a43b0245d423d4bf5b52ae3ea03_5"
          }
        },
        "ul":{
          "type":"object",
          "items":{
            "$ref":"#/definitions/8b503a43b0245d423d4bf5b52ae3ea03_5"
          }
        }
      }
    },
    "8b503a43b0245d423d4bf5b52ae3ea03_3":{
      "properties":{
        "ul_throughput":{
          "type":"number"
        },
        "dl_throughput":{
          "type":"number"
        },
        "imsi":{
          "type":"string"
        }
      }
    },
    "8b503a43b0245d423d4bf5b52ae3ea03_4":{
      "properties":{
        "avg_ul_throughput":{
          "type":"number"
        },
        "avg_dl_throughput":{
          "type":"number"
        },
        "imsi":{
          "type":"string"
        }
      }
    },
    "8b503a43b0245d423d4bf5b52ae3ea03_5":{
      "properties":{
        "duration_sec":{
          "type":"integer"
        },
        "table_id":{
          "type":"integer"
        },
        "priority":{
          "type":"integer"
        },
        "packet_count":{
          "type":"integer"
        },
        "byte_count":{
          "type":"integer"
        }
      }
    }
  }
}
```

Annex C QoE Optimizer control function OpenAPI specification

```
{
  "openapi": "3.0.0",
  "info": {
    "description": "This is the API exposed by the QoE_plugin",
    "version": "1.0.0",
    "title": "qoe_plugin_v1",
    "contact": {
      "email": "agraz@tsc.upc.edu"
    },
    "license": {
      "name": "Apache 2.0",
      "url": "http://www.apache.org/licenses/LICENSE-2.0.html"
    }
  },
  "servers": [
    {
      "url": "http://localhost:65005/"
    }
  ],
  "paths": {
    "/conf": {
      "post": {
        "summary": "Configure the parameters of the Slice",
        "description": "The operation is used to configure a set of parameters from the Management",
        "operationId": "confPlugin",
        "requestBody": {
          "content": {
            "application/json": {
              "schema": {
                "$ref": "#/components/schemas/QoEConf"
              }
            }
          }
        },
        "responses": {
          "200": {
            "description": "OK"
          },
          "400": {
            "description": "Bad Request"
          }
        }
      }
    },
    "/qoe": {
      "put": {
        "summary": "Set the perceived QoE (VERY BAD, BAD, FAIR, GOOD, VERY GOOD, EXCELLENT)",
        "description": "The operation is used to request an increase of the QoE if needed",
        "operationId": "send-QoE-Feedback",
        "requestBody": {
          "content": {
            "application/json": {
              "schema": {
                "$ref": "#/components/schemas/QoEObject"
              }
            }
          }
        },
        "responses": {
          "200": {
            "description": "OK"
          },
          "400": {
            "description": "Bad Request"
          }
        }
      }
    },
    "/qoe/{id}": {
      "get": {
        "summary": "Get the perceived QoE",
        "description": "The operation is used to request the last perceived QoE",
        "operationId": "get-Last-QoE-Feedback",
        "parameters": [
          {
            "name": "id",
            "in": "path",
            "description": "Id of the QoE Instance",
            "required": true,
            "schema": {
              "type": "string"
            }
          }
        ],
        "responses": {
          "200": {
            "description": "OK",
            "content": {
              "text/plain": {
                "schema": {
                  "type": "string"
                },
                "example": "10"
              }
            }
          }
        }
      }
    }
  },
  "components": {
    "schemas": {
      "QoEObject": {
        "type": "object",
        "properties": {
          "id": {
            "type": "string"
          },
          "value": {
            "type": "string"
          }
        }
      },
      "QoEConf": {
        "type": "object",
        "properties": {
          "slice_id": {
            "type": "string"
          },
          "slice_owner": {
            "type": "string"
          },
          "register_ip": {
            "type": "string"
          },
          "register_port": {
            "type": "string"
          }
        }
      }
    }
  }
}
```

Annex D UE control plugin OpenAPI specification

```
{
  "swagger":"2.0",
  "info":{
    "description":"The API of the UE control plugin",
    "version":"0.1",
    "title":"plug_and_play_ue_ctrl_plugin",
    "contact":{
      "name":"Giacomo Bernini",
      "email":"g.bernini@nextworks.it"
    },
    "license":{
      "name":"Apache License Version 2.0",
      "url":"http://www.apache.org/licenses/LICENSE-2.0"
    }
  },
  "host":"localhost:65006",
  "basePath":"/",
  "tags":[
    {
      "name":"config-rest-controller",
      "description":"Config Rest Controller"
    },
    {
      "name":"plugin-rest-controller",
      "description":"Plugin Rest Controller"
    }
  ],
  "paths":{
    "/plugin/control/ues":{
      "post":{
        "tags":[
          "plugin-rest-controller"
        ],
        "summary":"add new UE to slice",
        "operationId":"add-New-UE-to-slice",
        "consumes":[
          "application/json"
        ],
        "produces":[
          "*/*"
        ],
        "parameters":[
          {
            "in":"body",
            "name":"request",
            "description":"request",
            "required":true,
            "schema":{
              "$ref":"#/definitions/Ue"
            }
          }
        ],
        "responses":{
          "200":{
            "description":"OK",
            "schema":{
              "type":"string"
            }
          },
          "201":{
            "description":"Created",
            "schema":{
              "type":"string"
            }
          },
          "400":{
            "description":"Bad Request"
          },
          "401":{
            "description":"Unauthorized"
          },
          "403":{
            "description":"Forbidden"
          },
          "404":{
            "description":"Not Found"
          }
        }
      }
    },
    "/plugin/control/ues/{ueId}":{
      "get":{
        "tags":[
          "plugin-rest-controller"
        ],
        "summary":"get UE info",
        "operationId":"get-UE-info",
        "consumes":[
          "application/json"
        ],
        "produces":[
          "*/*"
        ],
        "parameters":[
          {
            "name":"ueId",
            "in":"path",
            "description":"ueId",
            "required":true,
            "type":"string"
          }
        ],
        "responses":{
          "200":{
            "description":"Success",
            "schema":{
              "$ref":"#/definitions/UeInfo"
            }
          },
          "401":{
            "description":"Unauthorized"
          },
          "403":{
            "description":"Forbidden"
          },
          "404":{
            "description":"Not Found"
          }
        }
      }
    },
    "/plugin/management/config":{
      "post":{
        "tags":[
          "config-rest-controller"
        ],
        "summary":"setConfig",
        "operationId":"confPlugin",
        "consumes":[
          "application/json"
        ],
        "produces":[
          "*/*"
        ],
        "parameters":[
          {
            "in":"body",
            "name":"config",
            "description":"config",
            "required":true,
            "schema":{
              "$ref":"#/definitions/PluginConfig"
            }
          }
        ],
        "responses":{
          "200":{
            "description":"OK",
            "schema":{
              "type":"string"
            }
          },
          "201":{
            "description":"Created",
            "schema":{
              "type":"string"
            }
          },
          "400":{
            "description":"Bad Request"
          },
          "401":{
            "description":"Unauthorized"
          },
          "403":{
            "description":"Forbidden"
          },
          "404":{
            "description":"Not Found"
          }
        }
      }
    }
  },
  "definitions":{
    "PluginConfig":{
      "type":"object",
      "properties":{
        "register_ip":{
          "type":"string"
        },
        "register_port":{
          "type":"string"
        },
        "slice_id":{
          "type":"string"
        },
        "slice_owner":{
          "type":"string"
        }
      }
    },
    "UeInfo":{
      "type":"object",
      "properties":{
        "imsi":{
          "type":"string"
        },
        "ip":{
          "type":"string"
        },
        "location":{
          "type":"string"
        },
        "status":{
          "type":"string"
        }
      }
    },
    "Ue":{
      "type":"object",
      "properties":{
        "imsi":{
          "type":"string"
        }
      }
    }
  }
}
```

Annex E QoS monitoring control function OpenAPI specification

```
{
  "swagger": "2.0",
  "info": {
    "description": "This is the API exposed by QoS Plugin",
    "version": "1.0.0",
    "title": "qos_plugin",
    "contact": {
      "email": "p.giardina@nextworks.it"
    },
    "license": {
      "name": "Apache 2.0",
      "url": "http://www.apache.org/licenses/LICENSE-2.0.html"
    }
  },
  "basePath": "/localhost:65007",
  "tags": [
    {
      "name": "qos_p",
      "description": "Everything about qos_plugin"
    }
  ],
  "schemes": [
    "http"
  ],
  "paths": {
    "/qos_plugin/get_avg_throughput_info": {
      "get": {
        "tags": [
          "qos_p"
        ],
        "summary": "Get throughput informations",
        "description": "",
        "operationId": "get-Slice-Performances",
         "produces": [
          "application/json"
        ],
        "responses": {
          "200": {
            "description": "OK"
          }
        }
      }
    },
    "/cpsr-info": {
      "post":{
        "tags": [
          "qos_p"
        ],
        "summary": "Post CPSR info",
        "description": "",
        "operationId": "confPlugin",
        "produces": [
          "application/json"
        ],
        "responses": {
          "200": {
            "description": "successful operation"
          }
        }
      }
    }
  }
}
```