5G European Validation platform for Extensive trials

Deliverable D3.5
Final implementation of the interworking reference model

Project Details

Call	H2020-ICT-17-2018
Type of Action	RIA
Project start date	01/07/2018
Duration	36 months
GA No	815074

Deliverable Details

Deliverable WP:	WP3
Deliverable Task:	Task T3.3
Deliverable Identifier:	5G_EVE_D3.5
Deliverable Title:	Final implementation of the interworking reference model
Editor(s):	Marc Mollà Roselló
Author(s):	Ramón Perez (TELC); M. Femminella, F. Lombardo, S. Salsano, N. Blefari Melazzi, G.Reali (CNIT); Giacomo Bernini, Leonardo Agueci (NXW); Marc Mollà Roselló (ERI-ES); Jaime Garcia-Reinoso, Pablo Serrano Yañez-Mingot (UC3M); Grzegorz Panek, Michał Grzesik (ORA-PL);
Reviewer(s):	Giada Landi (NXW), Kostas Trichias (WINGS)
Contractual Date of Delivery:	31/05/2021
Submission Date:	31/05/2021
Dissemination Level:	PU
Status:	Final
Version:	1.0
File Name:	5G_EVE_D3.5

Deliverable History

Version	Date	Modification	Modified by
V0.1	*16/02/2021*	*First draft*	*Marc Mollà Roselló*
V0.2	*26/04/2021*	*Second draft*	*WP3*
V0.3	*04/05/2021*	*Draft for internal review*	*WP3*
V0.9	*24/05/2021*	*Final draft for QA review*	*WP3*
V0.10	*28/05/2021*	*QA review*	*Kostas Trichias (WINGS)*
V1.0	*28/05/2021*	*Final check*	*Mauro Boldi*

Table of Contents

List of Acronyms and Abbreviations

Acronym	*Meaning*
3GPP	Third Generation Partnership Project
5G	Fifth Generation
ACID	Atomicity, Consistency, Isolation and Durability
API	Application Programming Interface
DCI	Data Center Interconnection
ETSI	European Telecommunications Standards Institute
GUI	Graphical User Interface
HTTP	Hypertext Transfer Protocol
IP	Internet Protocol
IWF	Inter-Working Framework
IWL	Inter-Working Layer
LCM	Lifecycle Management
MANO	Management and Orchestration
MSNO	5G EVE Multi-Site Network Orchestrator
MSO-LO	5G EVE Multi-Site NSO to Local Orchestrator
NBI	North-Bound Interface
NFV	Network Function Virtualization
NFVO	NFV Orchestrator
NS	Network Service
NSD	Network Service Descriptor
NSO	Network Service Orchestrator
ONAP	Open Network Automation Platform
OSM	Open Source MANO
PNF	Physical Network Function
PNFD	PNF Descriptor
REST	Representational State Transfer (software architectural style)
SBI	South-Bound Interface
SQL	Structured Query Language
SSH	Secure Shell
TOSCA	Topology and Orchestration Specification for Cloud Applications
UML	Unified Modelling Language
VNF	Virtual Network Function
VNFD	VNF Descriptor
YAML	YAML Ain’t Mark-up Language

List of Figures

List of Tables

Executive Summary

This deliverable D3.5 is the final deliverable that describes the implementation and testing of the Interworking Layer (IWL). This deliverable contains the description of all IWL components, including design and features reported previously in D3.4 ([16]) and the new features developed in the last part of the 5G EVE project. With this approach, we want to offer a complete view of our work and a full detailed description of the Interworking Layer.

We include the low-level detail of each IWL component, explaining the offered services as well as the internal architecture. The most relevant feature included in the final version is the multi-site support, which allows to deploy an experiment across several 5G EVE sites. We explain how IWL implements a distributed transaction logic in order to perform the deployment of an experiment.

Also, we finish the integration of the IWL with all the 5G EVE sites, which implies the integration with different NFV-Os (ONAP and OSM) and the integration with Radio Controllers (EVER and NC).

We include a small update about the inter-site connectivity, with the successfully demonstrated Gaming use case, which required an inter-connection between Spain and Greece for the User Plane. Also, the final roadmap of the Inter Working Layer is included.

Finally, we include the results of pending tests that has been executed after the delivery of D3.7, mainly related to multi-site use cases and the integration with ONAP orchestrated sites.

1 Introduction

Deliverable D3.5 "*Final implementation of the interworking reference model*" contains the report of the Inter-Working Layer final software deliverables of the 5G EVE project, developed in the scope of Task 3.3 "*Interworking model implementation and deployment*". The deliverable contains the final public definitions of the Interworking Framework components' interfaces, the low-level description of the components as well as the references to the public repository for the software artefacts.

1.1 Initial context

This document is the last report of the Work Package 3 from 5G EVE project. It is based on the analysis, architecture and interface definitions included in D3.1 ([1]) and D3.2 ([2]). It is also a continuation of the work described in D3.3([3]) and D3.4 ([16]).

For the clarity of our description, we decided to do a full report of the WP3 work, and some parts might be also described in previous deliverables. We will highlight the content when that is happening.

1.2 Structure of the document

The main structure of this deliverable can be summarized as follows:

- Section 2 contains a summary of the Interworking Framework design that is included in the D3.1, D3.2 and D3.3. It also contains the description of the new requirements and components included in the IWL since the delivery of D3.3.
- Section 3 contains the description of the software included in this deliverable. We included the public specification of the API provided for each component, with a reference of the public open API definition. Also, we included the features provided by each component as well as the internal design.
- Section 4 contains an update of the status of the inter-site connectivity.
- Section 5 describes the Interworking Framework roadmap at the time of this delivery.
- Section 6 contains the pending tests from D3.7 included in this deliverable.

2 Interworking Framework design

The IWL is a core component of the 5G EVE platform as it allows to abstract the specific site facility capabilities and offered services through a common and unified data model, enabling the design and execution of multi-site vertical experiments at the 5G EVE portal level through a common approach irrespectively of the specific technology constraints that each site implements to provide 5G services.

In practice, the IWL abstracts the heterogeneous capabilities of each site enabling the implementation of a unified end-to-end 5G experimentation and validation facility. At the same time, it guarantees the seamless interoperability of the services offered by the different site facilities through multi-site integration and delivery procedures. To achieve this, at its southbound the IWL is equipped with a set of adaptation functionalities (part of the IWL Adaptation Layer) that provide the required abstraction on top of the site facilities, harmonizing under a common interworking model and set of APIs (offered to the internal IWL logics) the heterogeneous capabilities offered by each site in terms of service and slice orchestration, RAN control, functions and services catalogues, monitoring and runtime configuration. Similarly, at its northbound the IWL offers a set of Interworking APIs that leverage on the common interworking model to expose towards the 5G EVE Portal unified and technology-independent discovery of per-site available services and VNFs, and provisioning of slices and services in support of multi-site 5G vertical experiments. This way, the 5G EVE Portal can first retrieve the capabilities of the multi-site 5G EVE infrastructure and then request for the instantiation of multi-site 5G vertical experiments designed on top of such capabilities, all by using a common and unified set of Interworking APIs based on the ETSI NFV SOL specifications [21].

Deliverables D3.1 and D3.2 already provided the detailed specification of the IWL architecture, which has then been used as the reference baseline for its implementation in the form of software prototypes with deliverables D3.3 and D3.4. As this document provides the very final software release of the 5G EVE IWL, Figure 1 shows the reference IWL architecture, which is composed by the integration of six main functional components: the Multi-site Catalogue, the Multi-site Network Service Orchestrator, the Multi-site Inventory, the Runtime Configurator, the Data Collection Manager and the IWF Repository. The Adaptation Layer is the enabling southbound abstraction IWL functionality that provides a common access interface and interworking model to access the various site facilities services and resources. On the other hand, the Interworking APIs exposed by the IWL at its northbound are the collection of the northbound APIs implemented and exposed by each of the six internal components. The Multi-Site Catalogue is the central repository of the whole 5G EVE platform for what concerns the capabilities offered by the site facilities in terms of NFV Network Services and VNFs and that are exposed through a common model that follows the ETSI NFV SOL Network Service descriptors (NSDs) and VNF descriptors (VNFDs) specifications. The Multi-Site Network Orchestrator orchestrates the multi-domain Network Services that implement the multi-domain 5G vertical experiments and coordinates the various local orchestrators in each site. It exposes to the 5G EVE portal Network Service Lifecycle Management APIs following the ETSI NFV SOL005 specifications. The Multi-Site Inventory maintains up-to-date information about instances of multi-domain and single-domain Network Services deployed in the 5G EVE site facilities, exposing such information to the 5G EVE Portal still following the ETSI NFV SOL005 model. The Data Collection Manager takes care to collect and persist all the required network and vertical performance metrics during the execution of experiments, following a publish/subscribe mechanism to consume such data. The Runtime Configurator is responsible for configuring the VNFs deployed as part of the multi-site Network Services in support of the vertical experiments, therefore providing so-called “Day-2” (i.e., after service and VNF instantiation) configuration APIs towards the 5G EVE Portal. Finally, the IWF Repository is the central storage point within the IWL and maintains all the site facility information that are useful for the internal IWL logics (e.g. related to site features, resources, credentials to access local orchestrators, etc.). It exposes dedicated APIs for the other IWL components to access such information.

The next subsection highlights the new features of the various IWL components which have been developed on top of the software prototypes delivered with deliverable D3.4, while the next chapter provides an overview of each IWL software component, in terms of software architecture, API specification (in the form of OpenAPI representations) and main service and internal logic description.

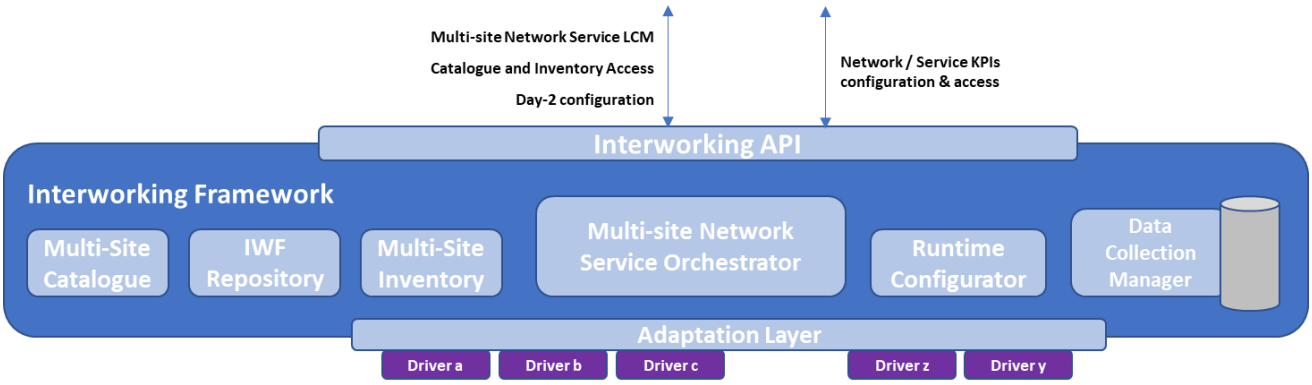

Figure 1: 5G EVE IWL final architecture

2.1 New features in D3.5 version

Multi-Site Catalogue

The Multi-Site Catalogue prototype released with this final software drop provides some new features with respect to the previous D3.4 version. In particular, new catalogue internal logics and procedures have been implemented to support the design and execution of multi-site 5G vertical experiments. Indeed, this final release includes support of composite NSDs to model multi-site Network Services. In addition, the support of NSD and VNFD runtime updates has been implemented, together with PNFD onboarding and removal when triggered from local-site orchestrators. Moreover, as part of the integration with local site and external orchestrators, the Multi-Site Catalogue has been equipped with new southbound drivers to enable the interaction with ETSI OSM R7 and R8, as well as with ONAP Frankfurt version. Dedicated support and integration have been also performed to interact with the ICT-19 5Growth platform.

Multi-site Orchestration

The Multi-Site Network Orchestration implements new logic for allowing the deployment of a Network Service across multiple 5G EVE sites. For that, the MSNO supports distributed transaction procedures that deploy or terminate in an atomic way all the components of a distributed Network Service.

Radio Slice Profile

In the final version, IWL supports the definition of Radio Slice parameters, where the experimenter can define requirements of latency, coverage, throughput or even the RAN technology to be used in the experiment. The IWL also provides new drivers for interacting with local NFV-O that are in charge of configuring the RAN, transport and core networks: EVER for Italian site and Network Controller for the Spanish site. Those new orchestrators are named as RAN-O (Radio Access Network Orchestrators)

3 Software artefacts

This section contains the description of the software artefacts included in the final delivery. For all Interworking framework components, we include the description of the public interface provided by the component, in Open API format. For the Multi-Site Catalogue, the Multi-Site Network Orchestrator and the Adaptation Layer we include a description of the services and features included in this deliverable. Additionally, we also include a low-level detail of the internal design of components.

3.1 Multi-Site Catalogue

This section provides an overview of the Multi-Site Catalogue software prototype. It is worth to mentioning that this document is the final release notes for the 5G EVE Interworking Framework software prototypes and for the sake of completeness of the reported information, this section includes the full list of features and functionalities implemented and supported by the Multi-Site Catalogue.

The Multi-Site Catalogue is the centralized repository of the whole 5G EVE platform in terms of single-site and multi-site capabilities offered by the site facilities. In practice, it stores all of the Network Service Descriptors (NSDs), VNF Descriptors (VNFDs) and PNF Descriptors (PNFDs) that can be used by the 5G EVE portal for the execution of vertical experiments. The information stored in the Multi-Site Catalogue is of two main categories:

i) NSDs, VNFDs and PNFDs collected directly from the site catalogues, as part of existing service and experiment capabilities on the related site facility,
ii) NSDs onboarded by the 5G EVE portal to model single-site and multi-site experiments, possibly composing NSDs, VNFDs and PNFDs already available.

Indeed, while VNFs and PNFs cannot be onboarded from the 5G EVE portal into the Multi-Site Catalogue (as they are considered pre-existing fundamental capabilities in the 5G EVE platform that shall be onboarded in the specific sites through a semi-automated procedure that pass through the 5G EVE Ticketing system), the 5G EVE portal can access in any moment its content to re-use existing and available site capabilities. For this, the Multi-Site Catalogue implements a continuous and periodic synchronization with the content of each site catalogue, with the aim of having up-to-date information stored, and consequently available for the 5G EVE portal and the other IWL components.

The Multi-Site Catalogue is aligned with ETSI NFV specifications, as it supports both APIs and related data models defined in the SOL specifications. In particular, the Multi-Site Catalogue implements part of the ETSI NFV SOL005 v2.4.1 APIs [5] at its northbound reference point (to cover NSD, VNFD and PNFD management operations), and it maintains the descriptors in accordance with the ETSI NFV SOL001 Tosca data model [8]. The main aim is therefore to expose a common (and standard) set of APIs towards the 5G EVE Portal and the other IWL components, while using a unified data model for describing the single-site and multi-site experiments available for deployment in the 5G EVE platform. To this aim, the Multi-Site Catalogue provides the required logic to translate the per-site specific NSD, VNFD and PNFD data models into the common one based on ETSI NFV SOL001. In the specific case of the 5G EVE site facilities catalogues, the translation logic is implemented for the ETSI OSM and ONAP descriptors data models.

In summary, with reference to the workflows described in deliverable D3.2, the Multi-site Catalogue implements the following Network Service, VNF and PNF on-boarding related procedures:

- *Catalogues synchronization*, to automatically retrieve, translate and store available NSDs, VNFDs and PNFDs from each site facility catalogue. This includes also the retrieval of updates to existing descriptors applied at runtime in the site catalogues;
- *VNF and PNF onboarding*, to dynamically and automatically retrieve, translate and store new VNFDs and PNFDs that are on-boarded directly in the 5G EVE sites catalogues;
- *VNF and PNF removal*, to dynamically and automatically remove existing VNFDs and PNFDs in the Multi-site Catalogue when the correspondent VNFDs and PNFDs are no longer available in the 5G EVE sites catalogues;

- *Network Service Descriptor onboarding triggered by the 5G EVE Portal*, to enable the 5G EVE Portal to upload in the IWL new NSDs for single-site and multi-site vertical experiments. While single-site related NSDs are translated and forwarded to the proper site catalogues into the specific data model format, the multi-site related NSDs are kept in the Multi-Site Catalogue only (and made available to the Multi-Site Network Orchestrator) in the form of composite NSDs (i.e. following the NSD nesting approach);
- *Network Service Descriptor onboarding triggered by the local site catalogues*, to dynamically and automatically retrieve, translate and store new NSDs that are on-boarded directly in the 5G EVE sites catalogues;
- *Network Service Descriptor removal triggered by the 5G EVE Portal*, to enable the 5G EVE Portal to remove existing NSDs whenever the related single-site or multi-site vertical experiment is no longer available in the 5G EVE platform for instantiation;
- *Network Service Descriptor removal triggered by the local site catalogues*, to dynamically and automatically remove existing NSDs in the Multi-site Catalogue when the related Network Services are no longer available in the 5G EVE sites catalogues.

3.1.1 Software architecture

The final version of the Multi-Site Catalogue software architecture is shown in Figure 2. In terms of design principles and main internal software components the approach already reported in previous deliverables is still applicable, as the internal architecture of the Multi-Site Catalogue has not required any major update. Indeed, as a final release note, the Multi-Site Catalogue is a Java application which uses PostgreSQL as its backend database to store NSDs, VNFDs and PNFDs information that is required for the core internal logics of the catalogue operations. On the other hand, the actual descriptor files (and CSAR software archives compliant with ETSI NFV SOL004 specification [10]) are stored in the local filesystem. In terms of software code, it is based on the 5G Apps and Services Catalogue that was developed in the context of the 5G-MEDIA project[1], which has been deeply evolved and enhanced to support the various 5G EVE single-site and multi-site vertical experiment preparation and execution procedures and workflows and made it suitable for use in the 5G EVE IWL.

As anticipated above, the Multi-Site Catalogue offers at its northbound reference point a subset of the ETSI NFV SOL005 REST APIs to cover the NSD and VNF Package management operations (i.e., for onboarding, updating, retrieving and deleting the various descriptors). The logic of the Multi-Site Catalogue is implemented in the Catalogue Service component, which glues the operations exposed towards the 5G EVE portal and the other IWL components with the actions to be taken on the site catalogues and the information that is stored locally. In practice, the coordination logic for the various Multi-Site Catalogue supported workflows are hosted in the Catalogue Service. A Kafka bus is used as part of the Multi-Site Catalogue operations for onboarding and updating NSDs, VNFDs and PNFDs in the IWL with the aim of decoupling the internal logic of the catalogue itself from the asynchronous interactions and translation logic of the site catalogue drivers at the southbound interface. In its final version, the Multi-Site Catalogue is equipped with two types of site catalogue drivers and is therefore able to interact with ETSI OSM (based on versions R3, R4, R6, R7 and R8, used in the Italian, Greek and Spanish site facilities) and ONAP (based on Dublin and Frankfurt versions, used in the French site). While the ETSI OSM driver implements a bi-directional translation logic (i.e., from ETSI NFV SOL001 Tosca model to ETSI OSM descriptors model and vice versa), for the ONAP driver the translation is only available in the bottom-up direction, i.e., to enable automated collection and retrieval of NSD and VNFDs from the ONAP catalogue. Each of these site catalogue drivers interacts with the IWF Repository to retrieve relevant information to access the given local site catalogues, in terms of credentials, tenants, URLs, endpoints, as well as available capabilities and automatically configure the driver itself.

[1] http://www.5gmedia.eu/

This allows, for each new per-site NFVO driver instance automatically created when the Multi-site Catalogue application starts, to retrieve the proper information to access the given local site catalogue (e.g., in terms of credentials, tenants, URLs, endpoints) and automatically configure the NFVO driver itself.

However, as depicted in Figure 2, a third driver exists for the Multi-Site Catalogue to interact with the ICT-19 5Growth platform [17]. While this driver has been developed in the context of the ICT-19 5Growth project [18] 5G EVE has supported the testing and integration activities to validate the interworking the 5Growth platform and vertical experiments.

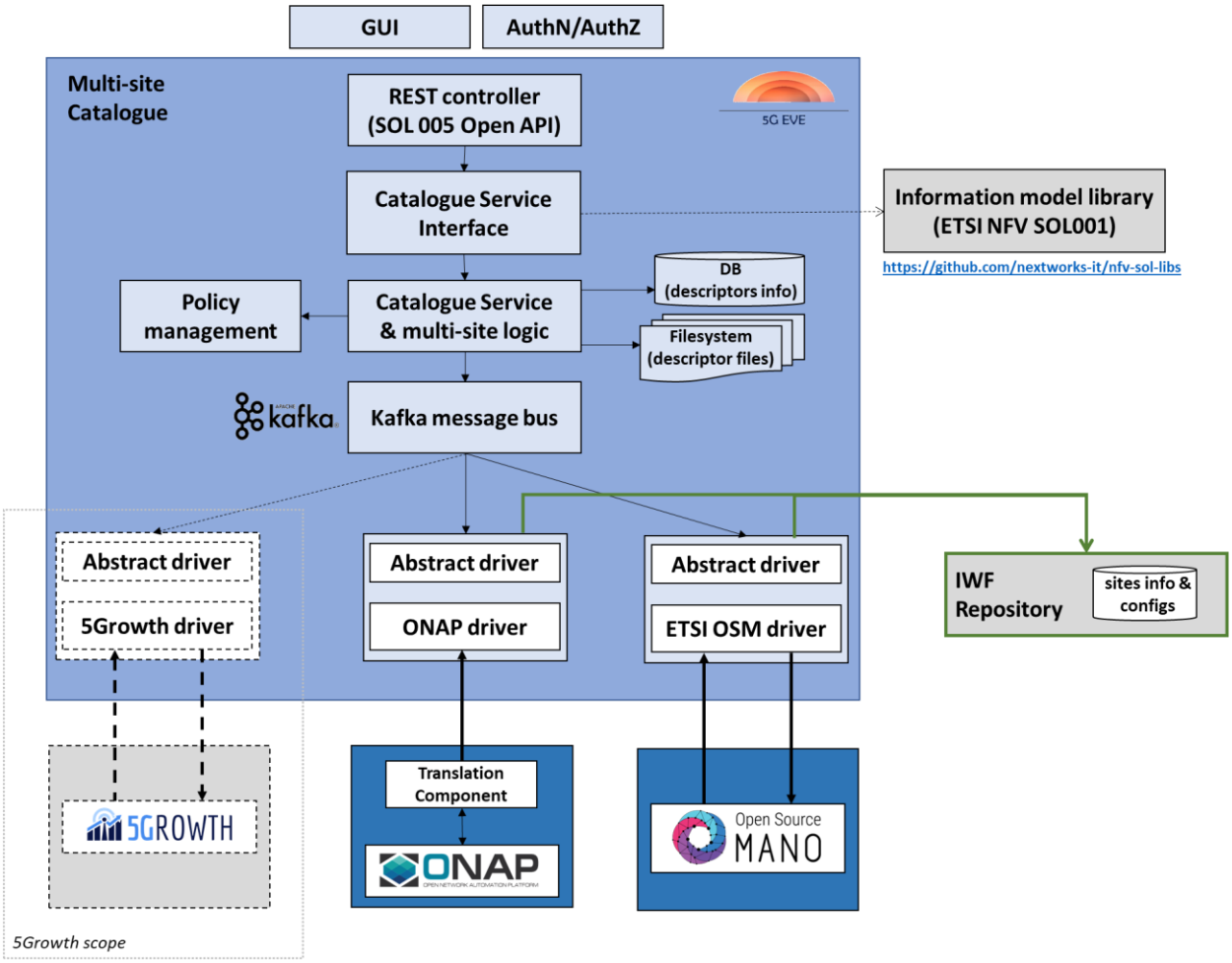

Figure 2: Multi-site Catalogue high level software design – final release version for D3.5

With respect to the last software release described in deliverable D3.4, beyond a consolidation of existing features and services as part of the multiple integration and validation activities carried out to support the various 5G EVE and ICT-19 vertical experiments, some additional features have been implemented as part of this final release. These can be summarized as follows:

- New Catalogue Service logics for the support of:
  - Composite NSD management, to model multi-site vertical experiments;
  - NSD update triggered by the 5G EVE portal, to dynamically modify;
  - NSD update triggered by the local site catalogues, to retrieve modifications to NSD;
  - VNFD update triggered by the local site catalogues;
  - PNFD retrieval from the local site catalogues;
  - PNFD removal triggered by the local site catalogues;
- Updates to the ETSI OSM driver to seamlessly support the latest R7 and R8 versions;
- Updates to the ONAP driver to seamlessly support the Frankfurt release deployed in the French site.

3.1.2 Open API description

As anticipated in the previous section, the Multi-Site Catalogue offers at its northbound reference point a set of REST APIs compliant with the ETSI NFV SOL005 v2.4.1 specification. The OpenAPI representations for the Multi-site Catalogue northbound REST APIs are available in the 5G EVE GitHub repository[2].

With respect to the last version reported in deliverable D3.4, the Multi-Site Catalogue has been equipped with an implementation of ETSI NFV SOL005 APIs for PNFD management, in particular to cover the query operations issued by the 5G EVE portal and other IWL components to retrieve available PNFs. Moreover, an explicit NSD update operation is also added in the form of a PATCH over an existing NSD content. Table 1 provides a summary of the whole set of APIs, related endpoints and main content exchanged as input/output that the final version of the Multi-Site Catalogue software prototype supports and implements. The last four rows of Table 1 list the new endpoints for NSD and PNFD management.

Table 1: Final list of services provided by Multi-Site Catalogue

Service	Path	Method	Input	Output	Description
Multi-Site Catalogue	**/ns_descriptors**	**POST**	**Create NsdInfo Request**	**NsdInfo Instance**	**Create a new NSD resource**
Multi-Site Catalogue	/ns_descriptors	GET	-	Array of NsdInfo	Returns the information of all NSD resources
Multi-Site Catalogue	/ns_descriptors/{nsdInfoId}	GET	-	NsdInfo	Returns the information of individual NSD resource
Multi-Site Catalogue	/ns_descriptors/{nsdInfoId}	DELETE	-	-	Delete individual NSD
Multi-Site Catalogue	/ns_descriptors/{nsdInfoId}/nsd_content	PUT	NSD file	-	Onboard new NSD content (TOSCA format)
Multi-Site Catalogue	/ns_descriptors/{nsdInfoId}/nsd_content	GET	-	NSD file	Returns the individual NSD content (TOSCA format)
Multi-Site Catalogue	/vnf_packages	GET	-	Array of VNFPckgInfo	Returns the information of all VNF Package resources
Multi-Site Catalogue	/vnf_packages/{vnfPkgId}/	GET	-	VNFPckgInfo	Returns the individual VNF Package resource
Multi-Site Catalogue	/vnf_packages/{vnfPkgId}/vnfd	GET	-	VNFD file	Returns the individual VNFD file (TOSCA format)

[2] https://github.com/5GEVE/OpenAPI/tree/v1.0/MultiSiteCatalogue

Multi-Site Catalogue	/ns_descriptors/{nsdInfoId}/nsd_content	PATCH	NSD file	-	Update existing NSD content (TOSCA format)
Multi-Site Catalogue	/pnf_descriptors	GET	-	Array of PnfdInfo	Returns the information of all PNFD resources
Multi-Site Catalogue	/pnf_descriptors/{pnfdInfoId}	GET	-	PnfdInfo	Returns the information of individual PNFD resource
Multi-Site Catalogue	/pnf_descriptors/{pnfdInfoId}/pnfd_content	GET	-	PNFD file	Returns the individual PNFD content (TOSCA format)

3.1.3 Service description

The version of the Multi-Site Catalogue software prototype released with this deliverable provides full support of the onboarding related workflows described in section 3.1, and can be considered as the final consolidated IWL catalogue in terms of features and functionalities as it went through a comprehensive testing, integration and validation processes during the preparation and execution of the 5G EVE and ICT-19 projects vertical experiments. However, beyond the regular fixes and improvements applied on top of the previous version released with deliverable D3.4, this final version of the Multi-Site Catalogue provides some new features (as anticipated in section 3.1.1), among which the most relevant and impacting ones (in terms of new offered functionalities) are those related to the new logics developed in the Catalogue Service for supporting NSD and VNFD updates and the support of composite NSDs to properly model and manage multi-site vertical experiment descriptors.

First of all, the dynamic NSD updates have been implemented to be supported in both directions, i.e. bottom-up (when triggered by a modification in the local site catalogue) and top-down (i.e. when triggered by an explicit update request from the 5G EVE portal at the northbound APIs of the Multi-Site Catalogue), while the VNFD updates only in the bottom-up one (as VNFs are not supposed to be either onboarded or updated through a fully automated procedure from the 5G EVE portal). In practice, the two bottom-up updates fall under the same workflow, as shown in Figure 3, where an explicit NSD or VNFD update from the local site administrator is detected by the Multi-Site Catalogue as part of the catalogues synchronization operation by adding an explicit additional step (step 5b in the figure) to verify if the given NSD or VNFD is changed with respect to the version stored locally (e.g. by inspecting either the NSD resource or VNF Package resource information or directly the descriptor content). In the case of NSD update through the 5G EVE portal, as depicted in Figure 4, the Multi-Site Catalogue, after having applied the regular sanity checks of the updated NSD content, verifies to which sites the given NSD was previously onboarded and perform in turn an update operation, after having translated the NSD content into the proper local site data model. According to the specific logic implemented by the given local site catalogue, an update operation (e.g., the one at step 5) could be implemented as a delete and re-onboard of updated NSD content.

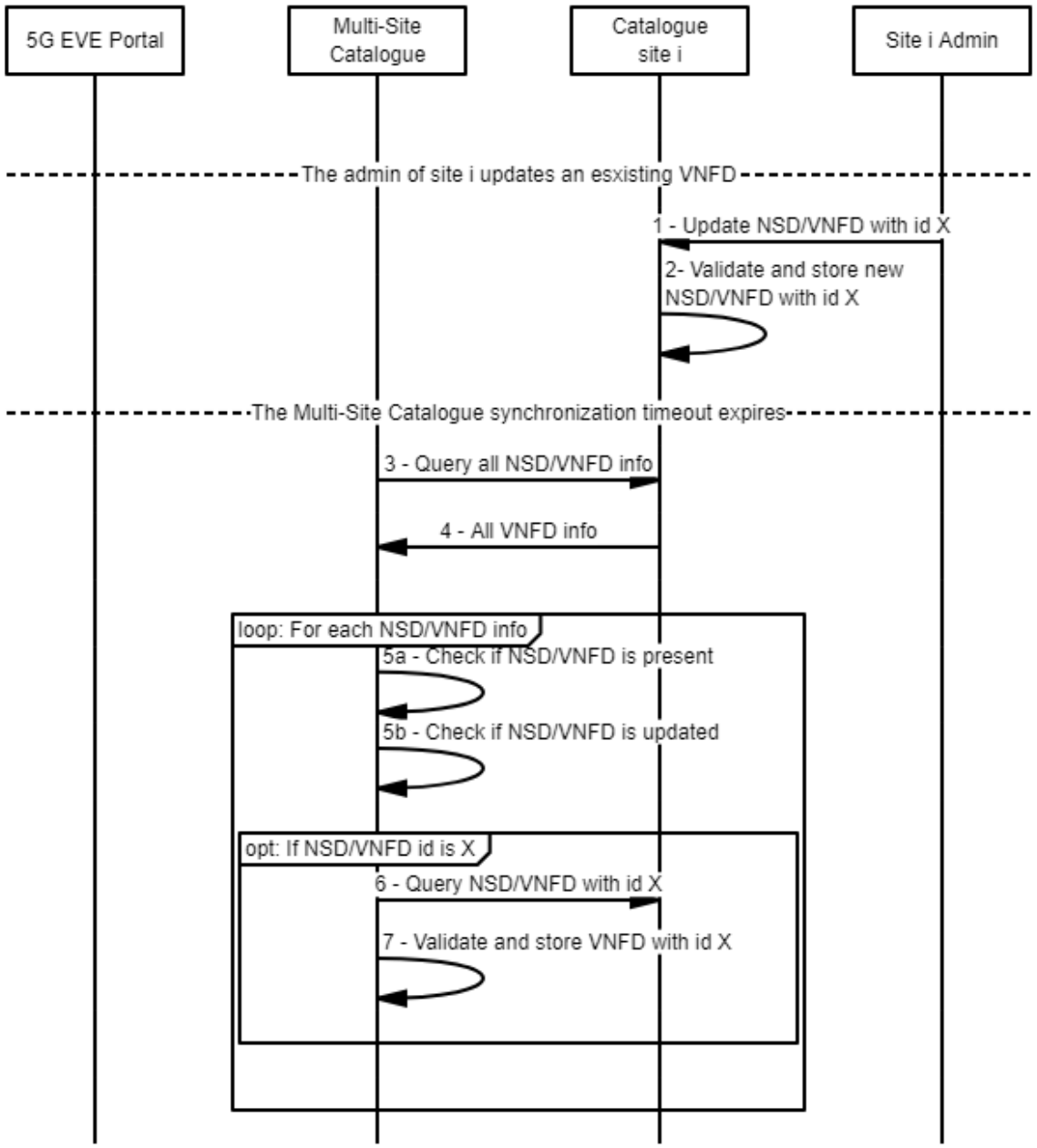

Figure 3: NSD and VNFD update triggered from local site catalogue.

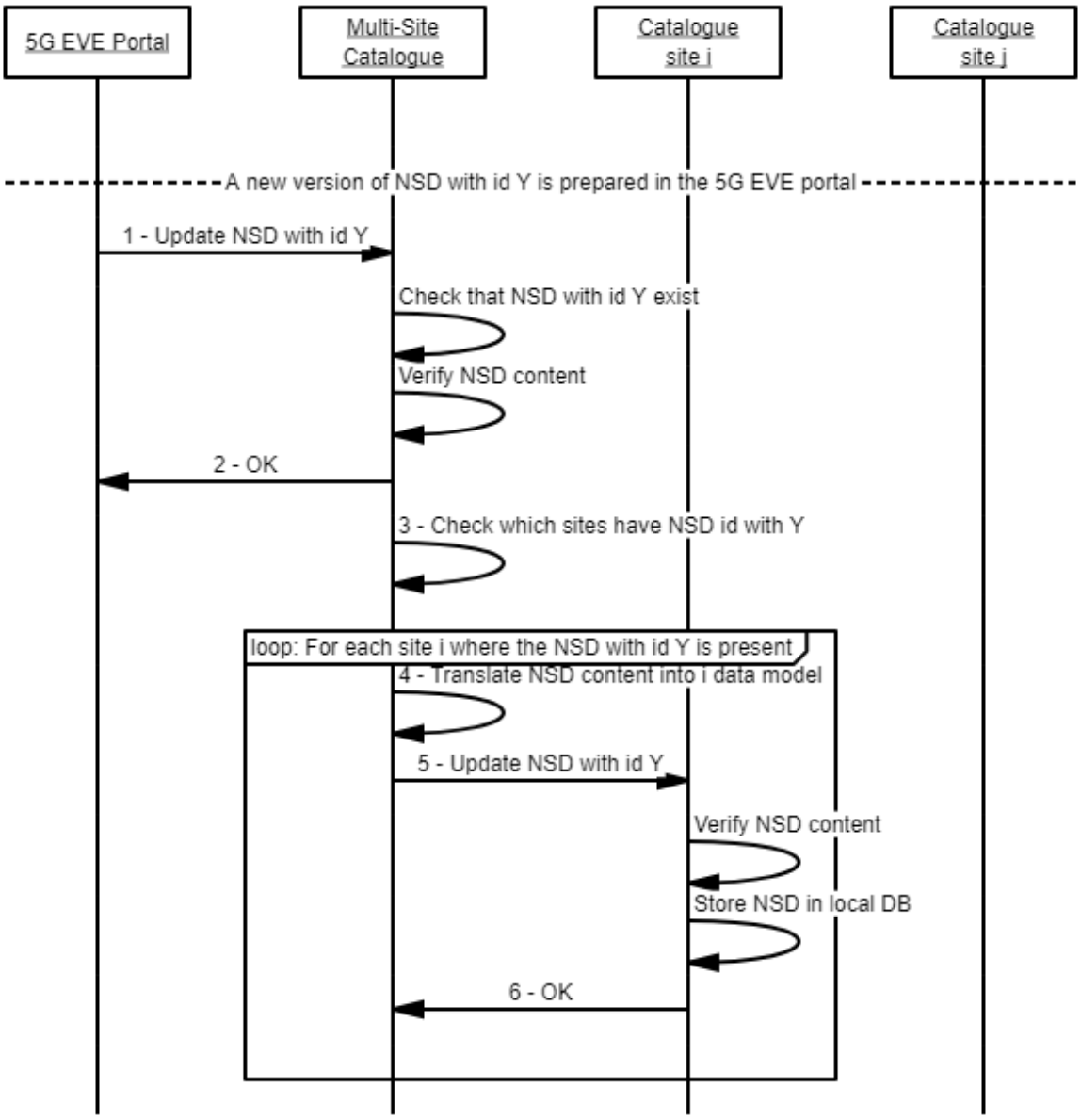

Figure 4: NSD update triggered from 5G EVE portal.

On the other hand, the management of multi-site vertical experiments as composite NSDs has required the Multi-Site Catalogue to provide support of the NSD nesting approach, which was not yet implemented up to the previous version. In practice, a composite NSD can be seen as a high level NSD that only reference other nested NSDs (at least two), as shown in Figure 5. In the 5G EVE scenario, such a composite NSD models a multi-site Network Service (i.e., a vertical experiment) that is composed by the concatenation of more Network Services (i.e., single-site parts of the end-to-end experiment), each to be instantiated and operated in a given site facility. The actual VNFDs are only included in the nested NSDs, while the composite NSDs contains the requirements in terms of virtual links and external connection points for the interconnection of the nested NSDs.

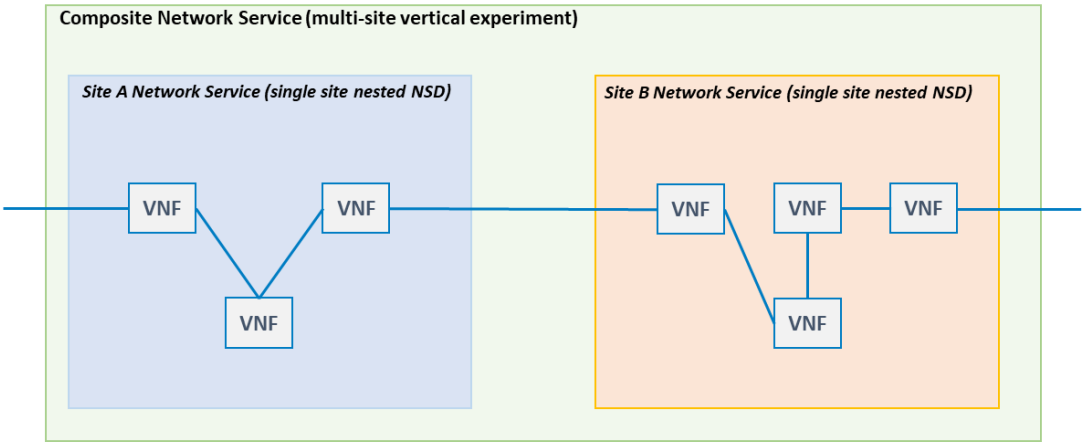

Figure 5: Multi-site experiment as composite NSD

From a practical implementation perspective, this final version of the Multi-Site Catalogue provides full support of the composite NSD onboarding workflow depicted in Figure 6. For this, the Catalogue Service logic has been enhanced to receive composite NSDs onboarding request from (only) the 5G EVE portal when a new multi-site experiment has to be uploaded in the IWL. As shown in the figure, the composite NSD is processed by the Multi-Site Catalogue to first check if all of the nested NSDs have been already onboarded in the catalogue and therefore are available in the local site catalogues (either through an explicit previous onboarding from the 5G EVE portal, or through a retrieval of the NSD from the local site catalogue itself). Indeed, as a pre-requisite, the nested NSDs shall be already stored in the Multi-Site Catalogue. Finally, the composite NSD is stored in the Multi-Site Catalogue only (i.e., it is not dispatched to any of the site catalogues) and it is kept therefore at disposal to the Multi-Site Network Orchestrator that will query it when the multi-site vertical experiment will be instantiated.

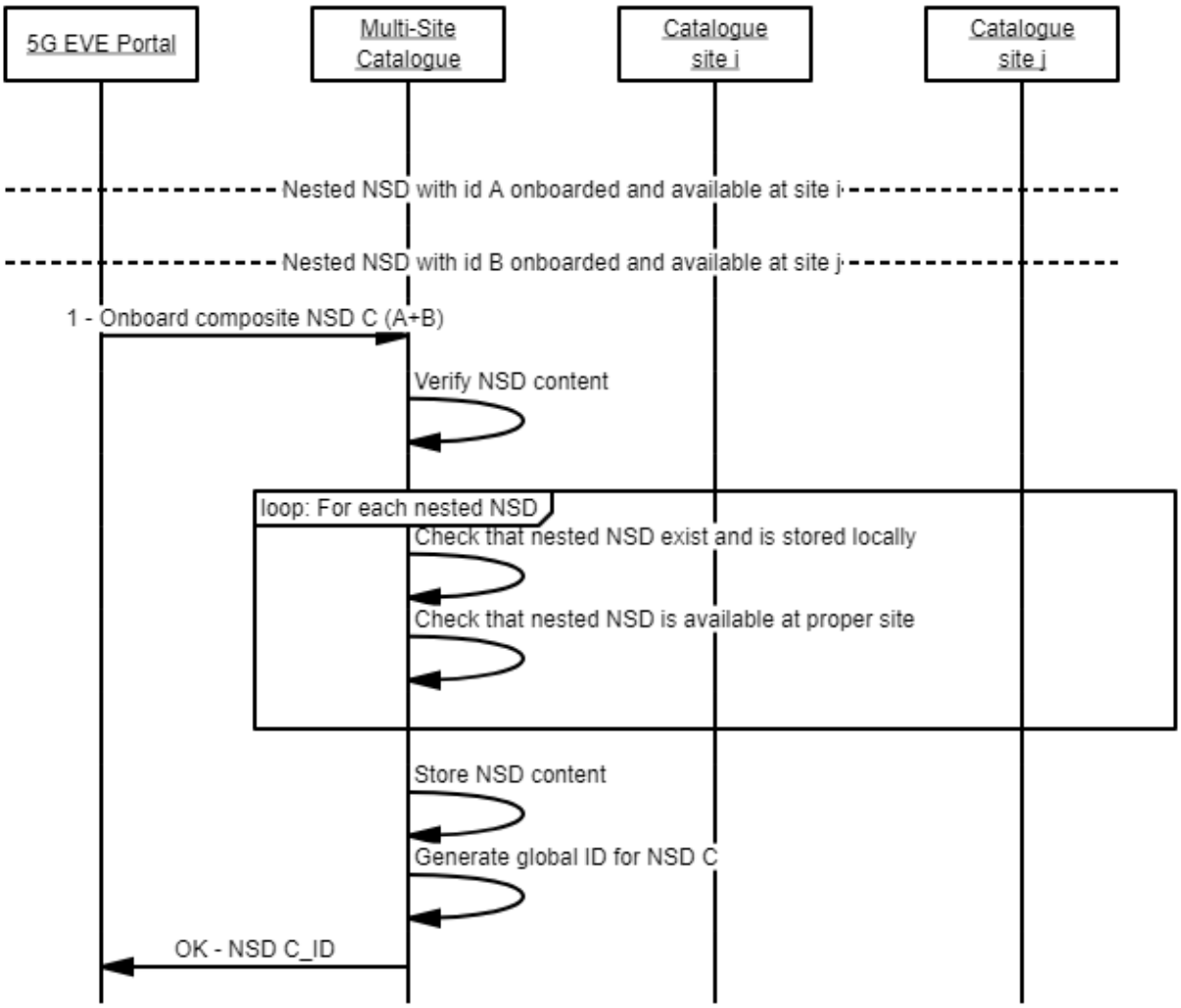

Figure 6: Composite NSD onboarding workflow

The consolidated version the Multi-site Catalogue delivered with this final software release is available as opensource code on the Nextworks GitHub[3].

A Docker Compose script is available to have a containerized deployment of the Multi-site Catalogue as the integration of three Docker containers:

i) one for the Multi-site Catalogue GUI front-end,
ii) one for the Multi-site Catalogue backend application,
iii) one for the Kafka message bus

The related Docker Compose script is available on the Nextworks GitHub[4].

3.1 Multi-Site Inventory

The Multi-Site Inventory is the component in charge of maintaining the status of the Network Services instantiated in any of the 5G EVE sites. As described in D3.2 [2], the Multi-site Inventory is fully managed by the Multi-site Network Service Orchestrator, which is in charge of notifying of all the changes that have to do with service provisioning.

[3] https://github.com/nextworks-it/5g-catalogue/tree/v4.2

[4] https://github.com/nextworks-it/5g-catalogue/tree/v4.2/deployments/docker

The new feature introduced in the last release is the support of multi-site Network Services, which are composed by a composite Network Service (root NS) with several nested NS associated to different 5G EVE sites. In order to avoid the exposure of this complex structure to the IWL clients, the Multi-Site Inventory aggregates the VNF information into the composite NS, providing a unified view of the NS status and information (e.g., IP addresses of the VNF). This extension is described in the Open-API of this component.

3.1.1 Open API description

The API of the Multi-Site Inventory can be found on the project Github[5].

The information included in the response of the NS Instance request are:

- Information about the Network Service, including the status.
- Information about the VNF(s) associated to the Network Service, including the status and the network interfaces (IP addresses).
- In case of multi-site Network Service, the information of VNF(s) are aggregated into the composite Network Service, including the information about the site.

The service includes a feature for translating the IP address of the VNF(s) to public addresses. This feature is configurable using the information stored at IWF repository.

In Table 2 we show the services provided by the Multi-Site Inventory.

Table 2: Services provided by Multi-Site Inventory

Service	Path	Method	Input	Output	Description
Multi-Site Inventory	/nslcm/v1/ns_instances	GET	-	Array of NsIntance	Returns the list of onboarded NS
Multi-Site Inventory	/nslcm/v1/ns_instances/{nsInstanceId}	GET	nsInstanceID	NsInstance	Retrieve the status of a NS

3.2 Multi-Site Network Orchestrator

The Multi-Site Network Orchestrator (MSNO) is one of the key components for deploying Network Services in the 5G EVE sites. As described in previous deliverables (please refer to [1], [2] and [3]), the MSNO offers a unique and standard API for managing the lifecycle of the Network Services in all the 5G EVE sites. The main mission of the MSNO is to unify the management of the Network Services, together with the Adaptation Layer, which provides a protocol conversion to per-site NFV-O APIs. It also interacts with other key IWL components like the Multi-Site Catalogue, from where MSNO obtains the NSD content and the NSD Information and with the IWL repository, which contains per-site information.

One of the most important features supported in the last release of the MSNO is the multi-site Network Service, which allows to deploy a NS in several 5G EVE sites. For that, the MSNO implements a distributed transaction logic that is described in the following sections.

Another important change is the support of the new RAN-O, in charge of orchestrating Radio resources. From MSNO point of view, the RAN-O are handled as other local orchestrator that provides a SOL 005 interface.

[5] https://github.com/5GEVE/OpenAPI/tree/v0.2/MSI

3.2.1 Software Architecture

The software architecture of the MSNO is based on a micro service architecture (Figure 7). We select this approach basically due to the flexibility that it provides during the development, especially with a small development team, as it allows to develop and to enhance each service individually. The main micro-service is the Multi-Site Network Orchestrator service, which implements the northbound interface that provides service to the 5G EVE Portal. We develop an auxiliary micro-service for the notifications part of the northbound interface, for decoupling the on-demand service to the Portal from the asynchronous operations. The MSNO service is also in charge of establishing the communications with the other IWL components.

All the micro-services use a storage service for the persistent storage, called storage service. The persistency is guarantee by using an open source reliable data base, called MongoDB[6].

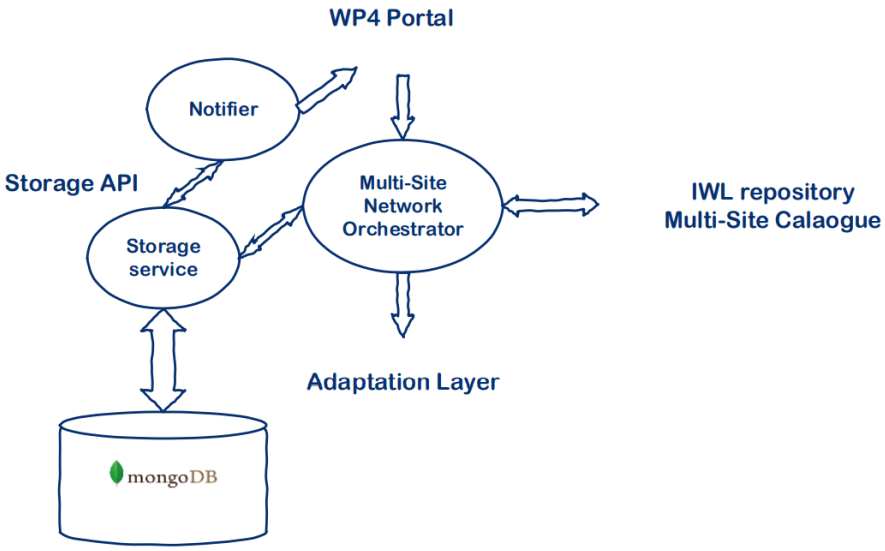

Figure 7: MSNO micro-service architecture

All the micro services share the same internal architecture, described in the following Figure 8. The development language selected for these processes is Go[7] and we use standard Go libraries for implementing the HTTP (net. HTTP), Service routing (gorilla.Mux) and YAML/JSON support.

[6] https://www.mongodb.com

[7] https://golang.org

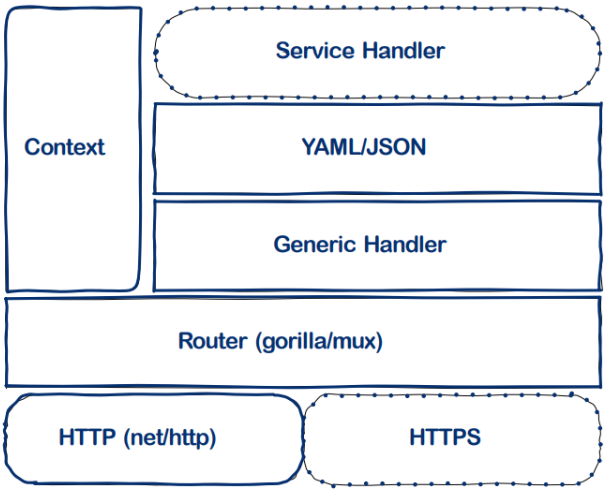

Figure 8: MSNO micro-service internal architecture

With this, we implemented a YAML parser in Python for automatically generating the basic code of each process from the OpenAPI specifications, which are described in the next section. This approach automatizes the common code of each micro-service and allows us to concentrate the effort in the development of the service logic. For the deployment of the micro services we selected the Docker[8] containers technology.

3.2.2 Open API description

The north-bound interface of the MSNO is available on the project Github[9].

It is based on the ETSI NFV SOL 005 ([5]) interface and implements a subset of the Network Service Life Cycle Management.

As commented, the service provided by the Multi-Site Network Orchestrator (MSNO) is an implementation of the ETSI NFV SOL 005 interface related to the Network Service Life Cycle Management.

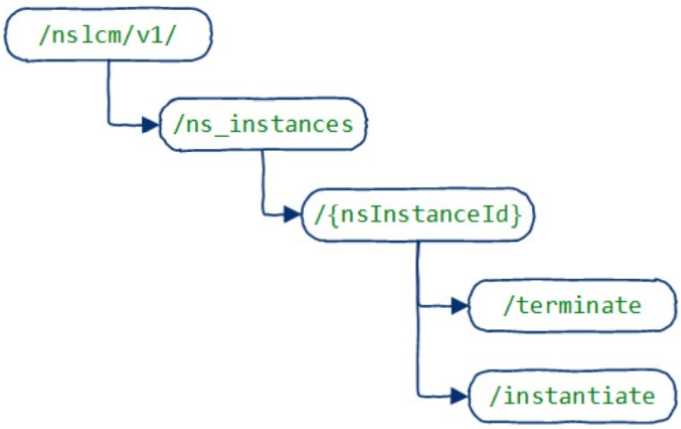

Figure 9: Multi-Site Network Orchestrator URI structure

[8] https://www.docker.com

[9] https://github.com/5GEVE/OpenAPI/tree/master/MSNO

Table 3: Service description of MSNO

Resource Name	Resource URI	HTTP Method	Description
NS Instances	/ns_instances[10]	GET	Returns the list of onboarded NS
NS Instances	/ns_instances	POST	Onboard a new NS
NS Instance	/ns_instances /{nsInstanceId}[11]	GET	Return the information of a NS
NS Instance	/ns_instances /{nsInstanceId}	DELETE	Deletes a NS
Instantiate NS	/ns_instances/nsInstanceId}/instantiate	POST	Instantiates a NS into target(s) site(s)
Terminate NS	/ns_instances/{nsInstanceId}/terminate	POST	Terminates an instantiated NS

3.2.3 Create Network Service Request

This request creates (onboard) a Network Service in the MSNO. It is the first step for deploying a Network Service into the Inter-Working Layer.

The MSNO does not raise any operation towards the 5G EVE sites as the target site is not known yet, so the NS remains in NOT_INSTANTIATE state in the MSNO.

The resource URI is: **{apiRoot}/nslcm/v1/ns_instances,** and the resource method is POST.

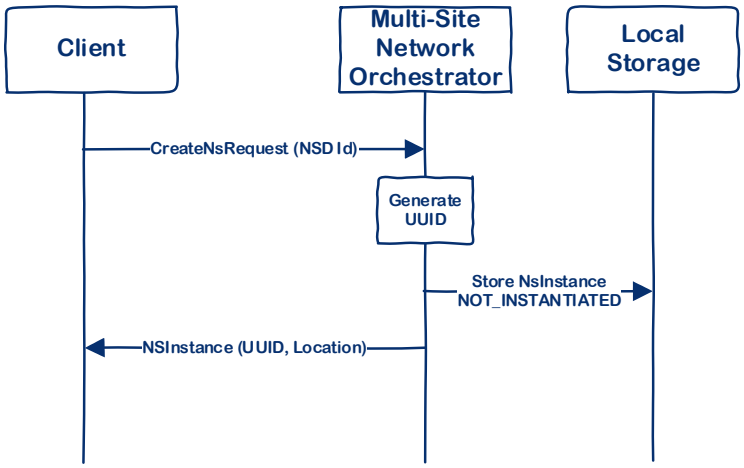

Figure 10: Create Network Service Workflow

The workflow is as following:

1. Client sends a CreateNSRequest with the NSD ID of the Network Service to be deployed;
2. MSNO generates a unique UUID for the NS Instance;
3. MSNO stores the NS instance in the local storage with the state NOT_INSTANTIATED;
4. Response includes the UUID for the NS and the link to the resource.

[10] Described in Multi-Site Inventory

[11] Described in Multi-Site Inventory

3.2.3.1 Query NS Instances

The Query Network Service Instances service returns the list of the Network Services that are onboarded in the Inter-Working Layer and its local status[12].

The resource URI is: **{apiRoot}/mslcm/v1/ns_instances**, and the resource method is GET.

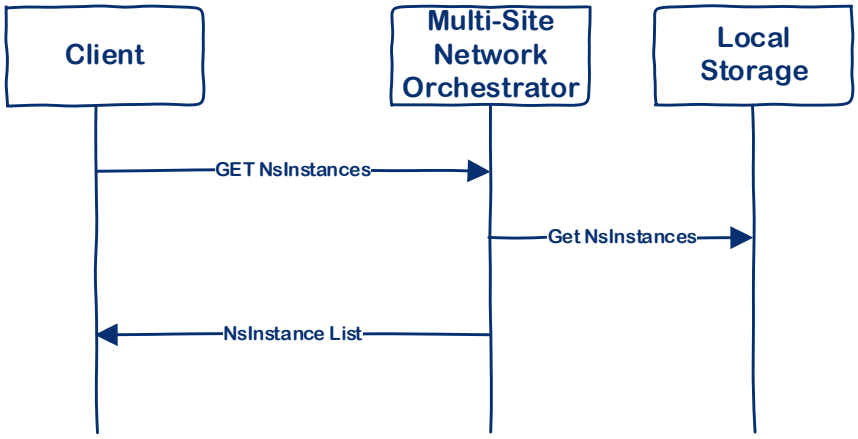

Figure 11: Workflow for the NS instances query

3.2.3.2 Query single-site NS Instance

This service allows to retrieve the status of a single Network Service Instance in the platform. MSNO will check with the site(s) NFV-O the last status of the NS before informing the client.

The resource URI is: **{apiRoot}/mslcm/v1/ns_instances/{nsInstanceId}**, and the resource method is GET.

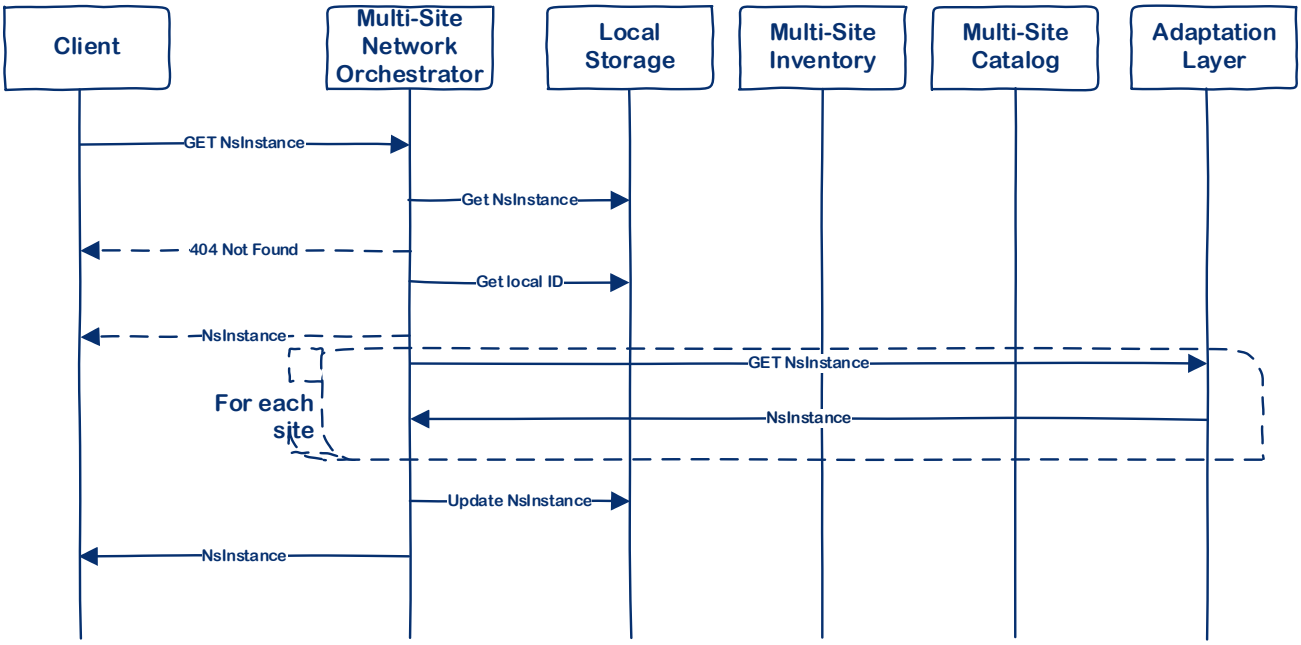

Figure 12: Multi-site query instance

The workflow is:

1. Client sends a GET request including the NS Instance;

[12] MSNO stores the last known state of the NS. In the list operation, the status of the NS is not updated, and it might be out-dated.

2. MSNO retrieves the NS Instance local information from storage;
3. MSNO retrieves the local NS ID from the mapping database, which allows to map with the site NS IDs;
4. In case that NS has not been instantiated yet, the MSNO returns the local information;
5. For each site with a nested NS, the MSNO retrieves the NS information from site NFV-O;
6. All the information is aggregated and stored locally;
7. The aggregated NS information is sent to the client.

3.2.3.3 Instantiate NS

This service allows to instantiate a Network Service that has been previously onboarded into the IWL. This is the second step for deploying a NS into 5G EVE as the result is the Network Service deploying in the target site(s) selected by the Experimenter.

The resource URI is: **{apiRoot}/mslcm/v1/ns_instances/{nsInstanceId}/instantiate**, and the resource method is POST.

In order to guarantee the backward compatibility of this operation with the previous releases (MS8 and MS9 releases), we implemented a selection logic described in Figure 13. The logic is as follows:

1. If SapData (Radio Slice Information) is not present and there is only one target site, the request is sent to the legacy logic (described in D3.4 [16])
2. If SapData is present, the new logic is used even if the NS is deployed in a single site

As it is mentioned, the MSNO implements a distributed transaction in order to deploy a Multi-Site Network Service. Each nested Network Service and Radio Slice is deployed in a single atomic transaction in parallel, and the MSNO is in charge of controlling the distributed transactions, as is described in Figure 14 and Figure 15.

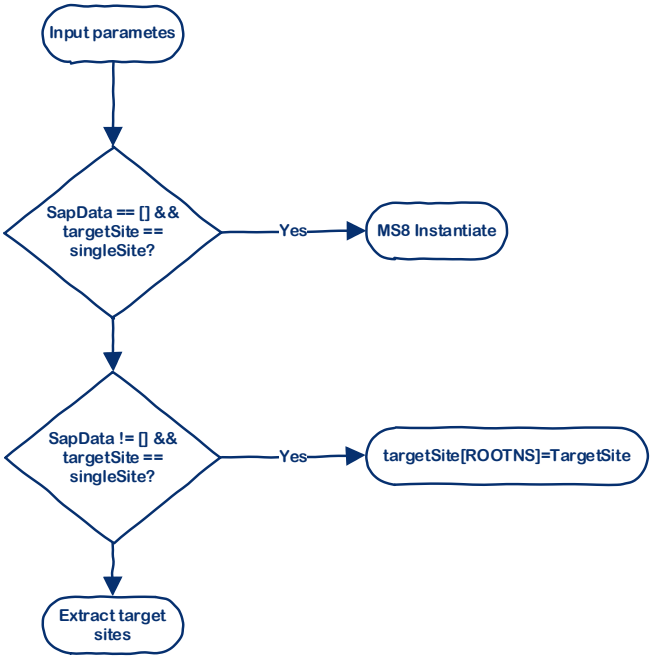

Figure 13: Selection logic

The workflow is as follows:

1. A Client send an Instantiation request with one or multiple nested NS and optionally with Radio Slice Information(s);
2. The MSNO analyse the input parameters as it is described in Figure 13;
3. The MSNO checks the local information to see if the NS is onboarded and not instantiated.

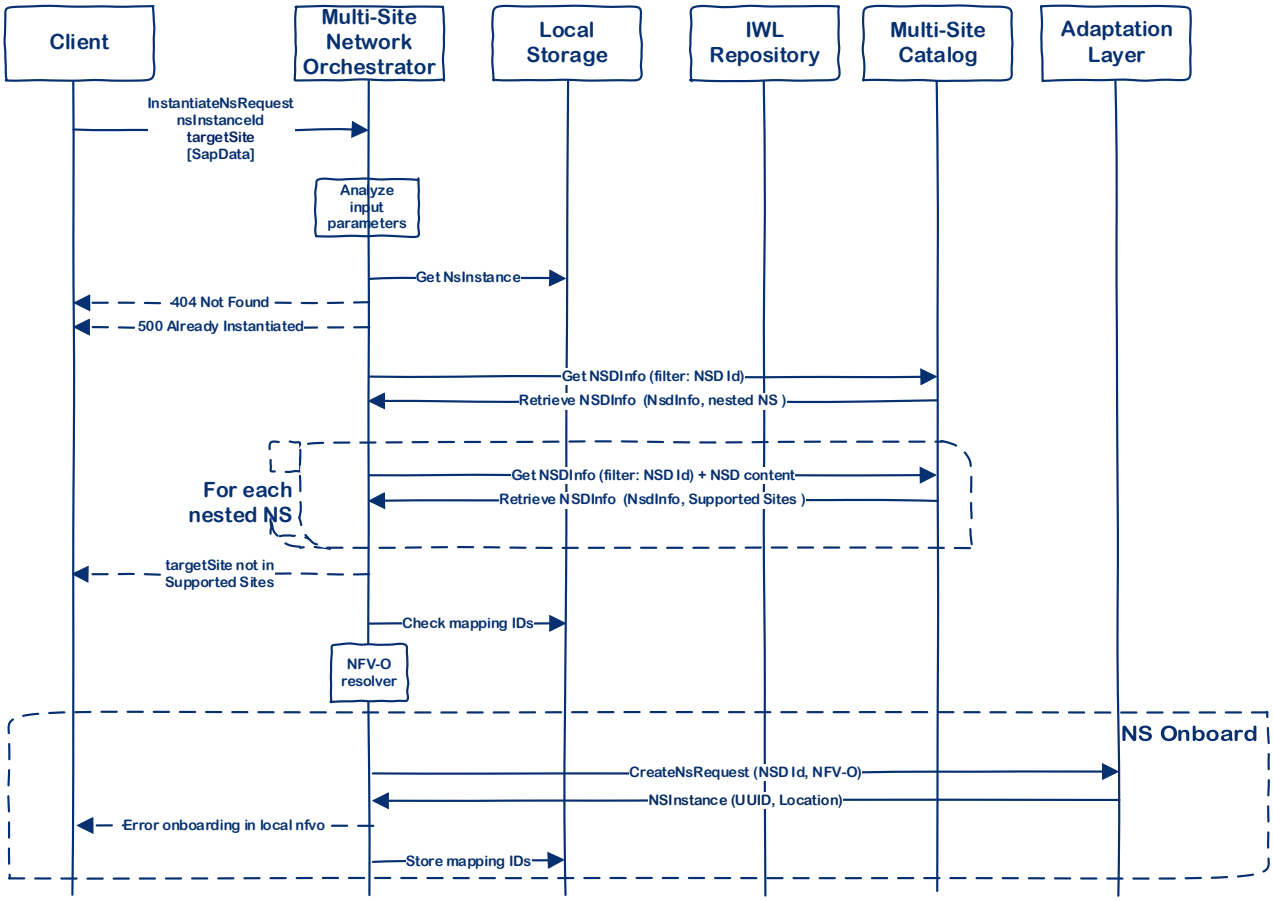

Figure 14: Workflow 1 for the Instantiate Request

In the next steps of the workflow, the MSNO interacts with the MS Catalogue to retrieve the NSD information of composite NSD and for each nested NSD. The MSNO also checks that the NSD is onboarded correctly in each of the sites.

Once the MSNO has all the information about composite and nested NSD, it resolves which NFV-O (for NS) and which RAN-O (for RAN) are involved in the distributed transaction and it starts in parallel the onboarding and instantiation process in each orchestrator as distributed atomic operations, as we can see in Figure 15.

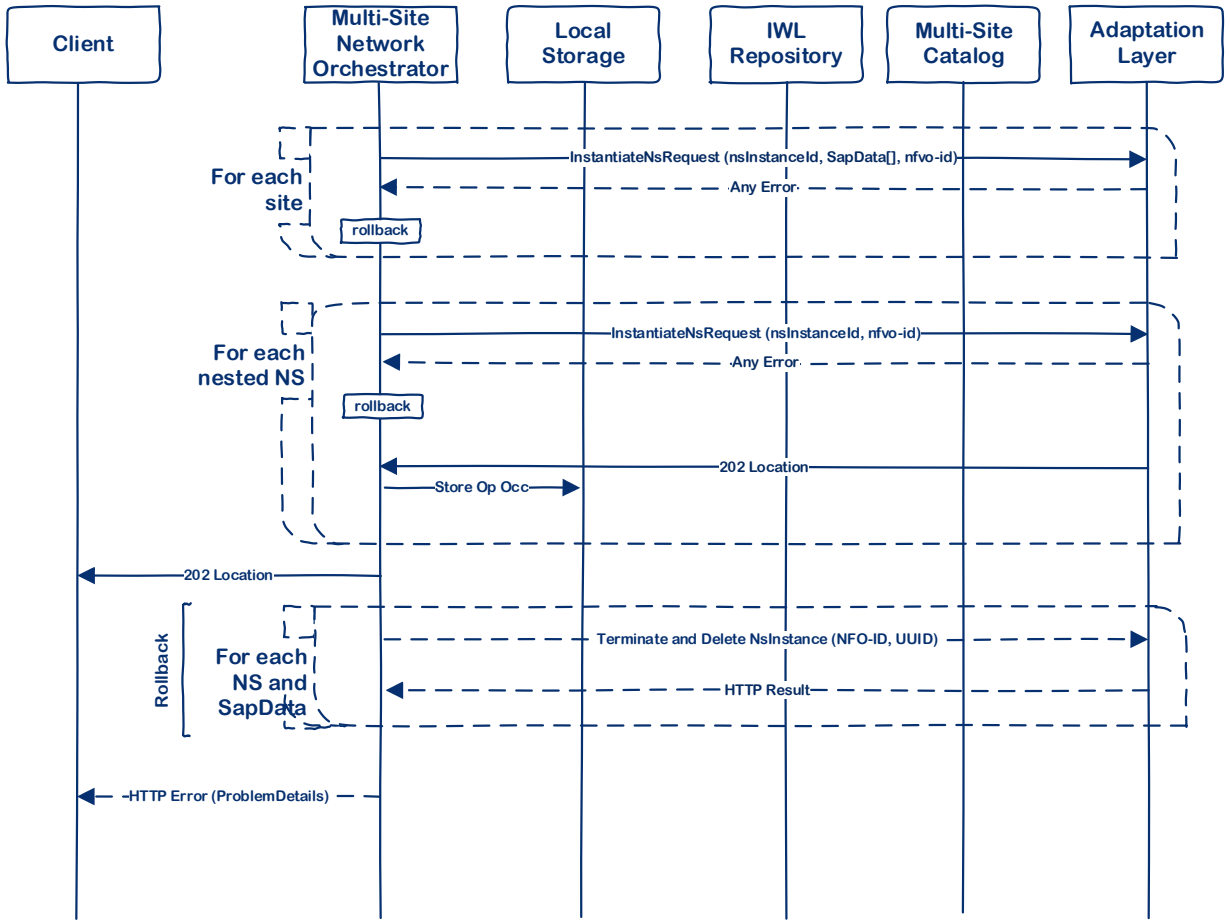

Figure 15: Workflow 2 of the Instantiation Request

In case of all transactions are successful, the MSNO returns the aggregated information of the composite NS, with a link to the new NS resource. In case of any issue, the MSNO has a distributed rollback procedure that is in charge of rolling back all the successful transactions, in order to remove any partial instantiation.

3.2.3.4 Terminate NS

This service allows to terminate a Network Service.

The resource URI is: **{apiRoot}/mslcm/v1/ns_instances/{nsInstanceId}/terminate**, and the resource method is POST.

The termination workflow implemented in MSNO is similar to the Instantiation workflow but in this case, the MSNO assumes that the distributed transaction will success in all NFV-O. In case of any failure, there is no special logic implemented in the MSNO as it is expected a manually intervention for checking the errors.

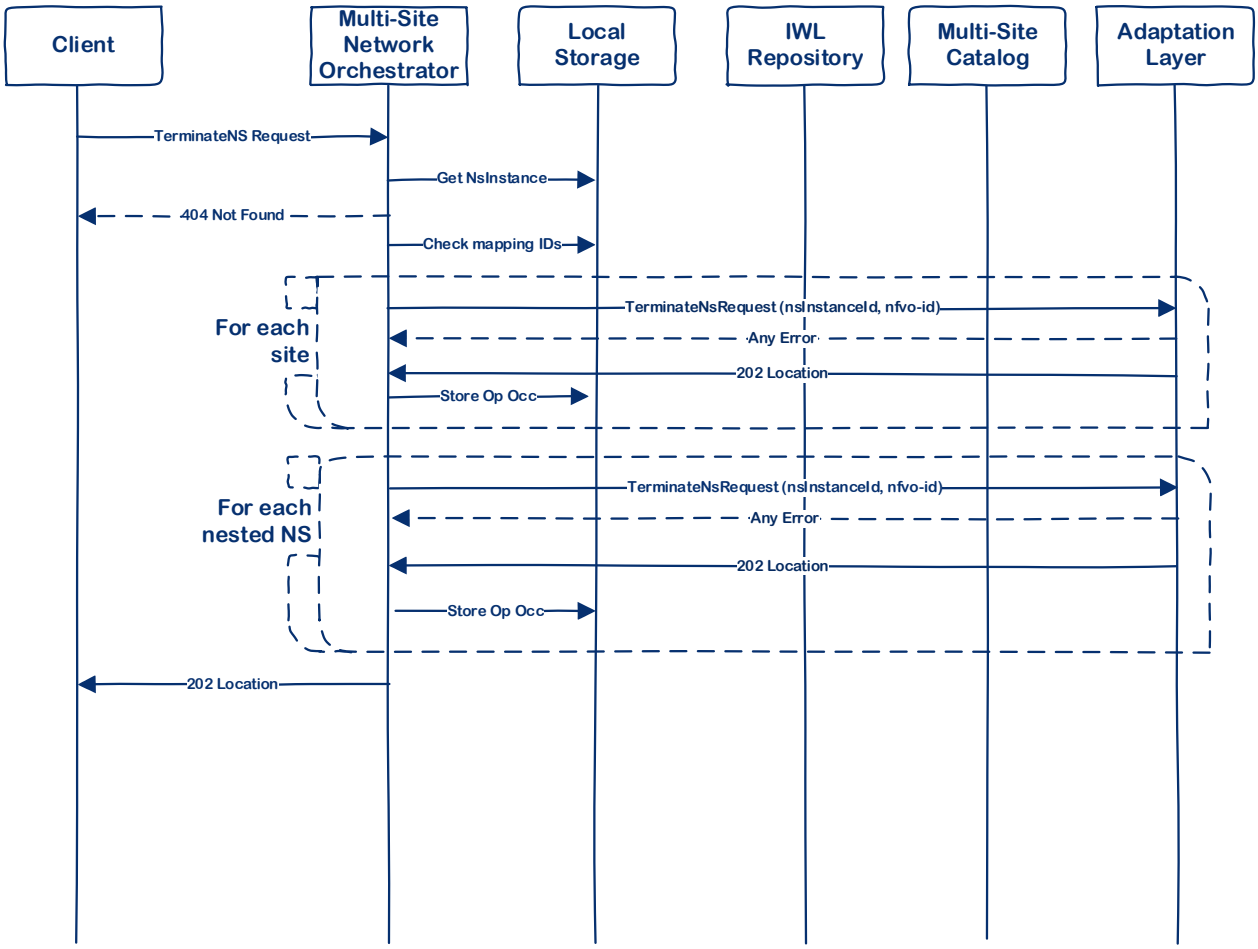

Figure 16: Termination Workflow

3.2.3.5 Delete NS

Deletes a Network Service from the 5G EVE platform. All resources are cleaned at IWL level as well as site NFV-O level.

The resource URI is: **{apiRoot}/mslcm/v1/ns_instances/{nsInstanceId}**, and the resource method is DELETE.

The workflow is:

1. Client sends a delete request with the NS instance ID;
2. MSNO retrieves local NS information;
3. In case of NS state is not NOT_INSTANTIATED, MSNO responds with a 409 Conflict (Note: as the local status in this drop is not synced with local NFVO, this check is skipped);
4. MSNO retrieves local ID for the NS instance;
5. MSNO sends the Delete NS request to the all NFV-O;
6. MSNO deletes NS and local ID from storage;
7. MSNO forwards the response to the client.

3.2.4 Docker HUB images

One of the improvements in the Continuous Integration that we implemented in the IWL is the delivery of the MSNO, that now is done using a public Docker HUB Image. The MSNO public images are available on the project repository[13][14].

3.3 Data Collection Manager

As already commented in the deliverable D3.4 ([16]), the Data Collection Manager (DCM) is the component responsible for the collection, persistence and delivery of all the network and vertical performance metrics, and also KPI values, which are required to be gathered during the execution of experiments, with two main objectives: monitor the experiment (thanks to the Monitoring and Result Collection tools provided within WP4 scope) and validate the targeted KPIs (thanks to the KPI Validation Framework proposed and developed within WP5 scope). To provide these capabilities, this component is based on the implementation of the publish-subscribe paradigm, using a message brokering system based on Apache Kafka[15] and Apache ZooKeeper[16].

The main improvement to be presented in this deliverable is the final configuration of the multi-broker architecture of the Data Collection Manager, enabling then the interconnection between the site brokers and the main broker placed in the Interworking Layer. This will be presented in the next subchapters.

3.3.1 Software architecture

To implement the multi-broker architecture for the Data Collection Manager, the system architecture has to be adapted to fully integrate the automatic configuration of all the brokers. This new, final architecture of the Data Collection Manager (DCM) is presented in the following Figure 17.

In this architecture, the names of the internal components of the DCM defined in deliverable D3.4 have been renamed to fit in a general design. In this way, the Python logic is now called DCM Handler, Apache ZooKeeper has been renamed to Broker Coordinator, and Apache Kafka is called Main Broker in the DCM and Site Broker in each site facility. Furthermore, the reference to "subscribe" operations in the diagram has been renamed to "deliver", as the real exchange of information between Kafka and the subscribers is the delivery of monitoring data.

The DCM Handler[17] performs the same functionalities reported in deliverable D3.4 (i.e., it is in charge of the topic management in coordination with the Experiment Lifecycle Manager (ELM) and with the Data Collection and Storage-Data Visualization (DCS-DV)), but also including new functionalities related to the new internal components included in the DCM architecture. These new components are the following:

- The DCM Site Plugin[18] is an entity running on each site facility, providing a REST API to manage the lifecycle of the topics to be created on each site. Therefore, the DCM Handler, apart from creating/deleting the topics in the DCM, it also sends a request to the DCM Site Plugin to create/delete the topic in the site facility.
- To coordinate the lifecycle of the topics to be created in the DCM and the sites, a set of MirrorMaker Processes (one for each topic related to metrics and KPIs instantiated in the system) are required. These

[13] https://hub.docker.com/repository/docker/5geve/msno

[14] https://hub.docker.com/repository/docker/5geve/storage

[15] https://kafka.apache.org/

[16] https://zookeeper.apache.org/

[17] Code available here: https://github.com/5GEVE/5geve-wp3-dcm-handler (tag v0.2).

[18] Code available here: https://github.com/5GEVE/5geve-wp3-dcm-site-plugin (tag v0.1).

processes, managed by the DCM Handler, are based on the Kafka's Mirroring feature[19], which enables the maintenance of a replica of an existing Kafka topic. In this way, each MirrorMaker Process implies the creation of a subscriber to the topic created in the site facility, together with a publisher that publishes the monitoring data received into the topic created in the DCM.

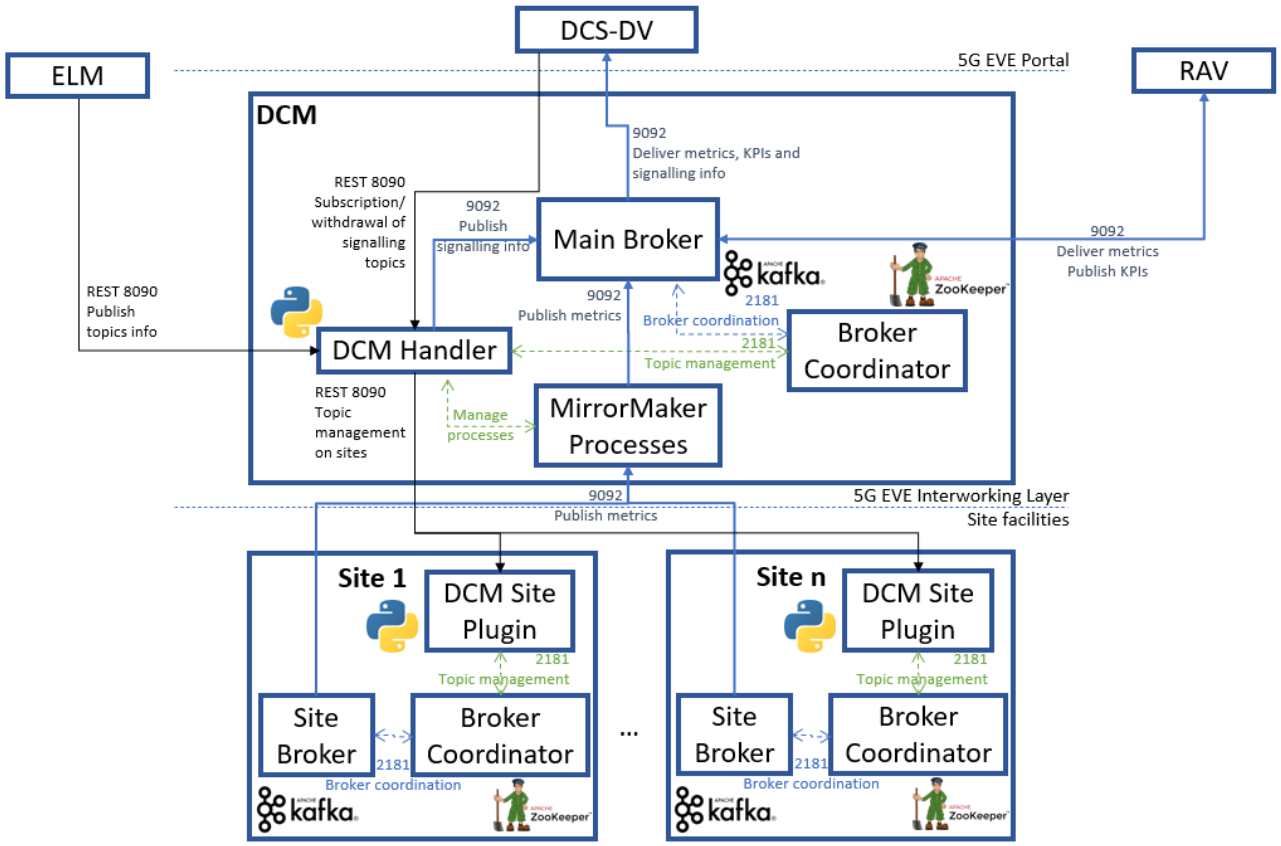

Figure 17: Data Collection Manager final architecture

Thanks to this new architecture, the separate management of each Kafka broker is assured, as each Kafka broker is coordinated by a single instance of the Broker Coordinator and is only accessed by the components connected to the site in which they are deployed (i.e. a component from a given site cannot access to the Kafka broker from a different site; to do this, it needs to connect to the Main Broker in the IWL, which has a copy of all the topics instantiated in the platform). This is one of the main changes in the architecture compared to the one presented in deliverable D3.4, in which one single Broker Coordinator was used to control all the deployed brokers. However, this design also enables the delivery of all the monitoring data towards the DCM in the Main Broker, offering this monitoring data to upper layers (i.e., the Result Analysis Validation (RAV) for calculating the KPIs associated, or the DCS-DV to save the data and display them through dashboards).

The ports used in the communication between components are the same than the ones presented in deliverable D3.4, also using the port 8090 in each DCM Site Plugin for the communication between the DCM Handler and this module. Its OpenAPI is also provided in Section 3.3.2.

Finally, note that the deployment steps to be followed to configure the DCM from scratch can be found in the 5G EVE Github repository[20].

[19] https://cwiki.apache.org/confluence/pages/viewpage.action?pageId=27846330

[20] Deployment steps available here: https://github.com/5GEVE/5geve-wp3-dcm-deployment (tag v0.2).

3.3.2 Open API description

The updated OpenAPI specifications of both the DCM Handler and the DCM Site Plugin are available in the 5G EVE Github repository[21], and are also presented in the following Table 4. In the case of the DCM Handler, it is the same than the one reported in deliverable D3.4.

Table 4: Data Collection Manager operations from its Open API specification

Service	Path	Method	Input	Output	Description
DCM Handler	/dcm/subscribe	POST	expId = internal topic = signalling topic name	201 – accepted 400 - error	Subscribe[22] to the signalling topic provided as input in the body request.
DCM Handler	/dcm/unsubscribe	DELETE	expId = internal topic = signalling topic name	201 – accepted 400 - error	Unsubscribe to the signalling topic provided as input in the body request.
DCM Handler	/dcm/publish/<topic>	POST	topic = signalling topic[23]. Array of records[24] containing values with the information related to the topics to be subscribed/unsubscribed.	201 – accepted 400 – error	Publish the information related to the topics that will be created during the experiment (and deleted afterwards) in the signalling topics, so that the DCM triggers all the mechanisms to create the topics in Kafka and to distribute them to the proper entities.
DCM Site Plugin	/dcm_plugin/<topic>	POST	topic to be created	200 – OK 500 - error	Create the topic provided in the URL in the site facility.
DCM Site Plugin	/dcm_plugin/<topic>	DELETE	topic to be deleted	200 – OK 500 - error	Delete the topic provided in the URL in the site facility.

3.3.3 Service description

The same service description provided for the DCM in deliverable D3.4 also applies to this last version of the component. However, there are some minor changes to be applied in the workflows to fit in the new multi-

[21] Specifications available here: https://github.com/5GEVE/OpenAPI/tree/master/DataCollectionManager.

[22] Note that the subscribe and unsubscribe operations for the topics related to metrics and KPIs do not need to expose an external API, as it is directly handled by the DCM Handler with direct interactions with Apache ZooKeeper.

[23] For the moment, the data managed by this endpoint is exclusively related to the messages sent by the ELM to subscribe/unsubscribe to the topics related to the experiment. This endpoint will not be finally used to publish data in Kafka in another topic.

[24] Examples with the format of this array of records can be found in deliverable D4.4 – Annex A.

broker architecture. As a result, the three phases related to the monitoring workflow are updated[25] in the following way:

- **Subscription phase (**Figure 18**)**: in the updated workflow, the interaction between the IWL and the sites to properly configure the topics is integrated. In this way, when the DCM Handler receives the topics to be created in the platform from the ELM (message 7'), it creates all the topics in the Main Broker (message 8'), and then it sends a request to the corresponding DCM Site Plugin for each topic (message 9'), which creates the same topic in the corresponding Site Broker and then confirms the operation (message 10') to the DCM Site Plugin. Finally, the DCM Site Plugin creates the MirrorMaker processes for each topic (message 11'), linking then the topics created in both brokers. After this, the data is delivered to the DCS (message 12').

[25] In this deliverable, the updates to be commented are the ones related to the Data Collection Manager. For the workflow related to the Portal components, please check the deliverable D4.6.

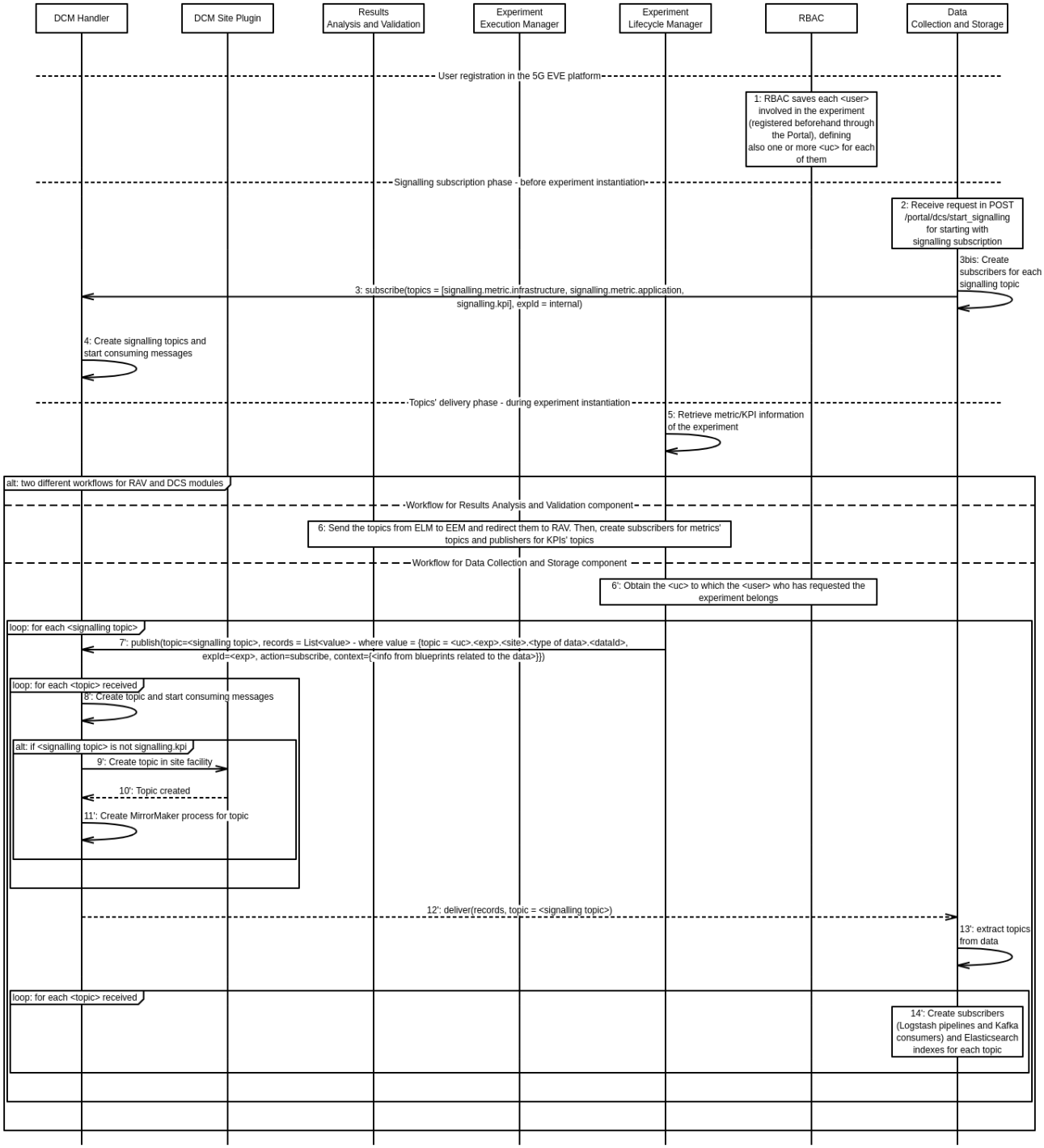

Figure 18: Subscription to the topics used for experiment monitoring and performance analysis purposes.

- **Monitoring and data collection phase (**Figure 19**):** in the same way than the previous workflow, the monitoring and data collection phase also presents the interaction between the sites and the Data Collection Manager. As a result, when the logs are extracted (message 1), transformed (message 2) and published (message 3) by the Data Shippers in the corresponding Site Broker, the monitoring data is then replicated towards the corresponding MirrorMaker process (message 4). As commented before, the MirrorMaker process, for each topic, has a subscriber for the topic created in the corresponding site facility and a publisher in the topic created in the DCM. So, when the subscriber receives the monitoring data, it automatically sends it to the Main Broker (message 5). After this, the workflow is exactly the same than the one presented in deliverable D3.4.

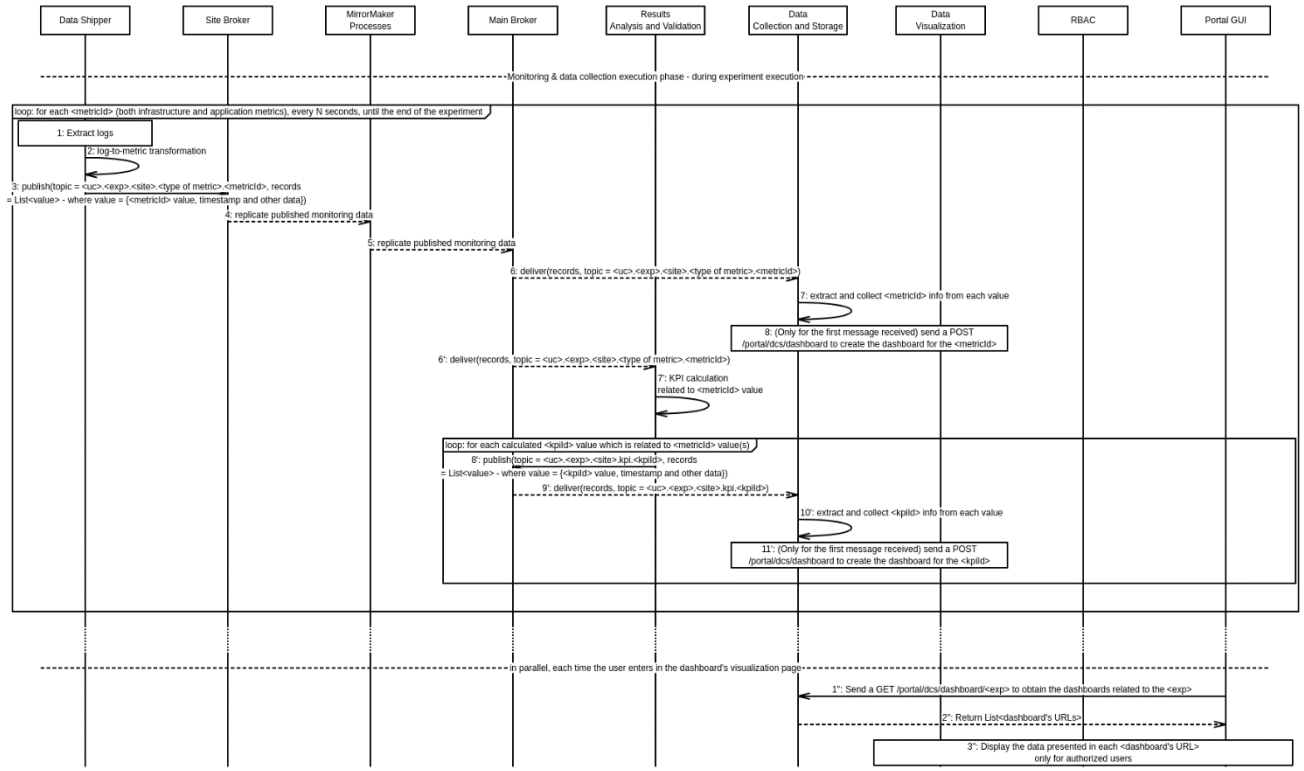

Figure 19: Delivery and management of monitoring information during the experiment execution

- **Withdrawal phase (**Figure 20**):** finally, the withdrawal phase follows an identical approach than the one presented for the subscription phase, but in the opposite way. Firstly, the configuration is removed from the DCS (messages from 3' to 5'), to remove the subscribers to the topics. Then, the MirrorMaker processes for each topic are deleted (message 6'), and after this, the DCM Handler contacts the corresponding DCM Site Plugin to delete the topic from the Site Broker (message 7'), receiving the confirmation from the DCM Site Plugin (message 8'). To conclude, the topic is also deleted from the Main Broker in the IWL (message 9').

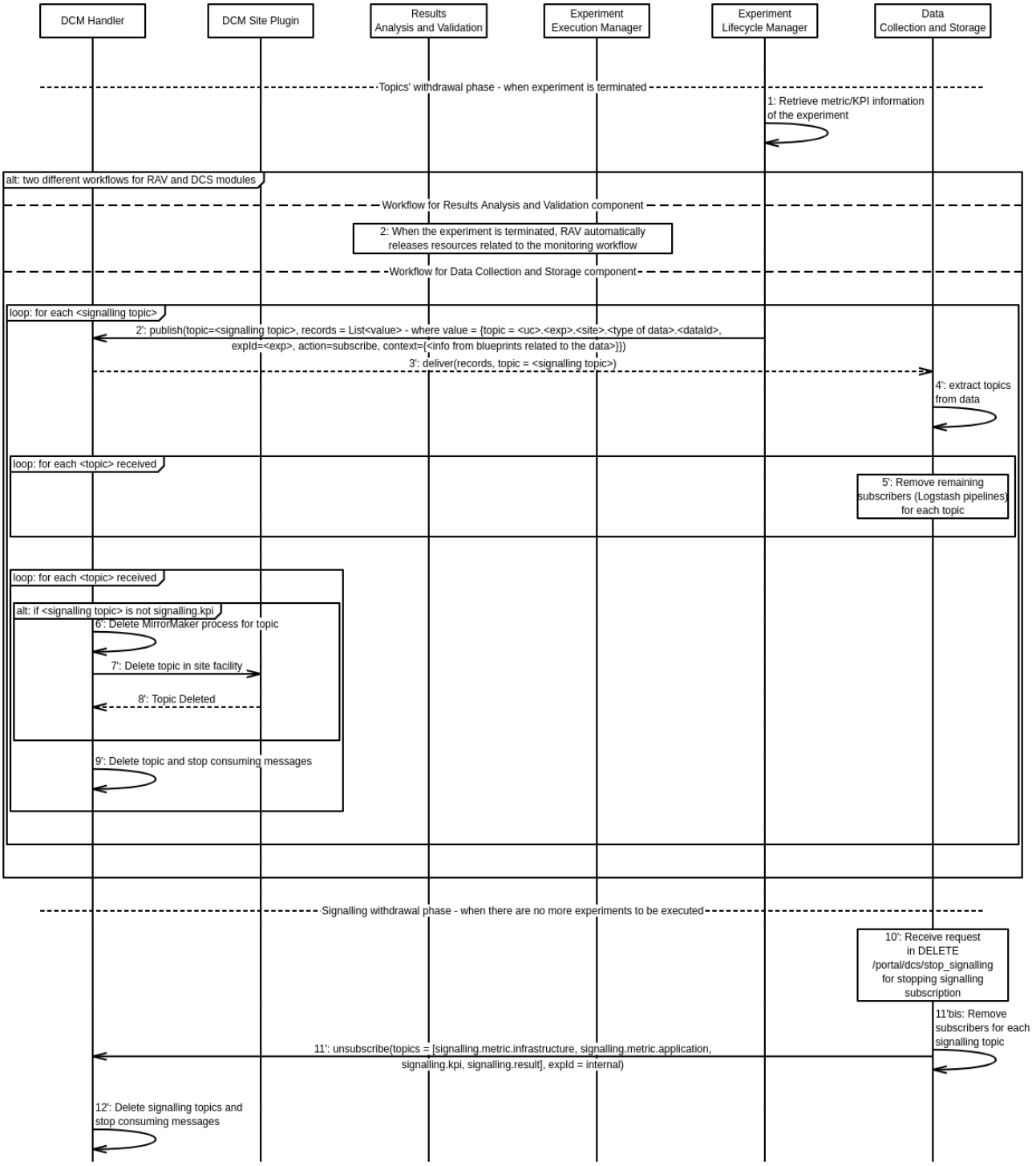

Figure 20: Withdrawal of the topics used for experiment monitoring and performance analysis purposes.

3.4 Runtime Configurator

As stated in deliverable D3.4, the Runtime Configurator (RC) is the component in charge of supporting the experiment configuration, execution and termination through the interaction with the Experiment Execution Manager (EEM), handling the commands to be executed in the different phases of an experiment, differentiating between the Day-2 configuration of the targeted servers, both VNFs/PNFs or infrastructure components, and the experiment execution itself, executing the commands related to each step of the experiment in the

corresponding servers. The main module of this component is Ansible Core[26], using SSH for connecting to the components from each site facility for executing the commands provided for a given execution. The interaction between the Experiment Execution Manager and the RC is based on a REST API, to easily manage the lifecycle of the configurations and executions to be made.

3.4.1 Software architecture

Compared to the architecture provided in deliverable D3.4, the proposed architecture of the RC is mainly the same but including small changes in the names used for the internal modules related to this component. The updated architecture is depicted in Figure 21.

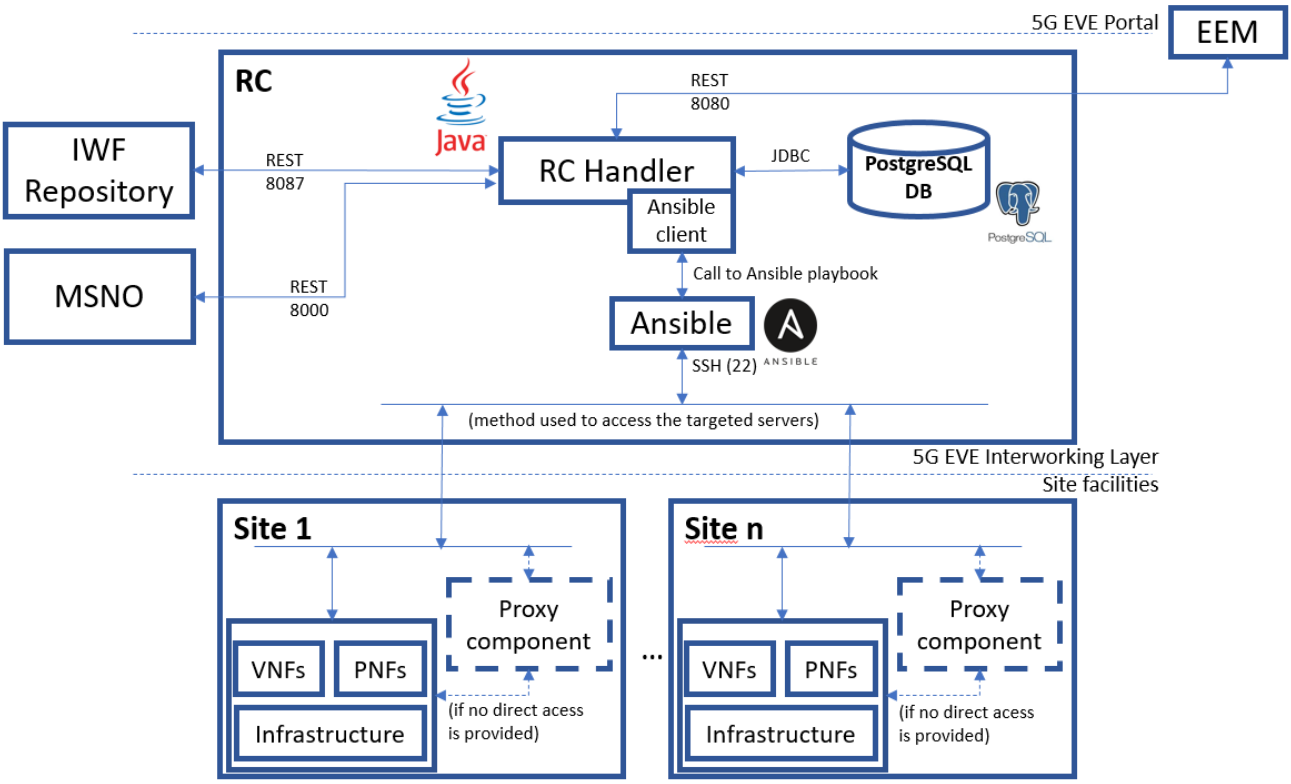

Figure 21: Runtime Configurator final architecture

Firstly, the Java logic has been renamed to RC Handler[27], to generalize the terms used to describe the modules. The features offered by the RC Handler have not changed, but it now includes interactions with the MSNO for retrieving the public/private IP addresses related to a given network service, to be used for certain Data Shippers related to infrastructure metrics that may need that information.

Another change that has been done is the removal of the REST client as a southbound API in the RC Handler, as all the use cases will finally interact by using Ansible[28].

As a result, there are small changes to be considered[29] in the data model managed by the RC Handler, which is presented in Figure 22.

[26] https://docs.ansible.com/ansible/latest/index.html

[27] Code available here: https://github.com/5GEVE/5geve-wp3-rc-handler (tag v0.3).

[28] The playbooks used by the RC Handler are available here: https://github.com/5GEVE/5geve-rc.

[29] The information not provided in this deliverable can be read in deliverable D3.4.

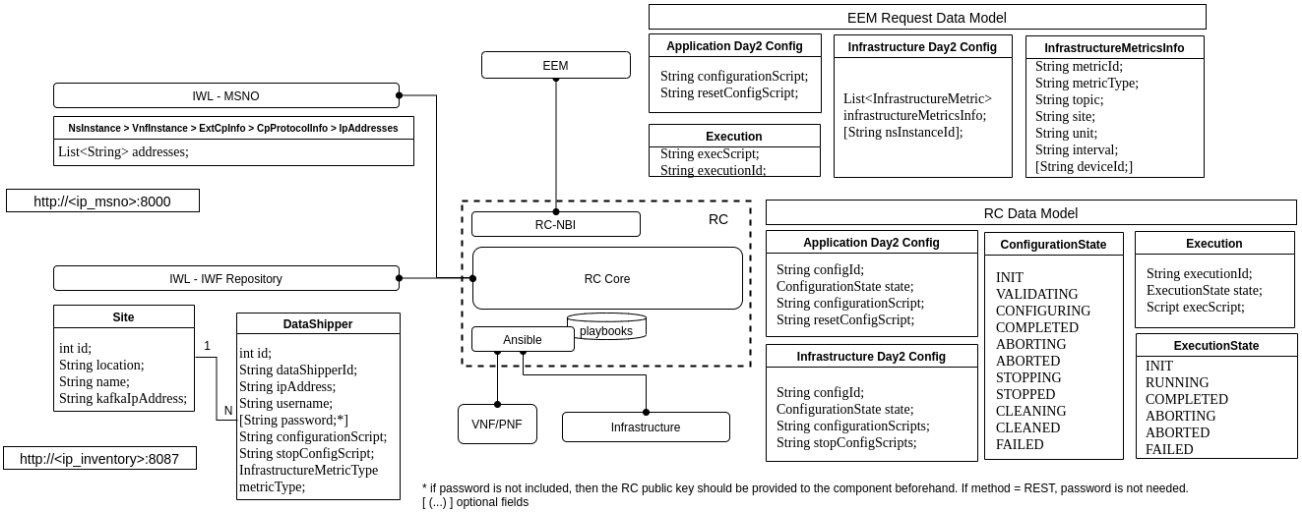

Figure 22: Runtime Configurator data model

The three types of request that are handled between the EEM and the RC are the same: (i) Application Day-2 configuration, (ii) Infrastructure Day-2 configuration and (iii) Execution. To fit in the new integration of the MSNO, the request made for the Infrastructure Day-2 configuration has been extended to include the nsInstanceId related to the experiment to be configured, which is optional. If provided, the RC will contact the MSNO to extract the list of IP addresses related to that network service.

Apart from that, the Script entity in the RC data model has been removed, using single strings to define the scripts to be saved on each entity. These strings are built based on the information obtained from the different sources involved in the RC workflow (i.e., EEM provides the scripts related to Application Day-2 configuration and Execution, and the IWF Repository provides the scripts related to Infrastructure Day-2 configuration).

Finally, as stated before, the access to the components deployed on site facilities will be only done by using Ansible, which uses the OpenSSH protocol by using specific Ansible playbooks.

In the same way that the DCM, note that the deployment steps to be followed to configure the RC from scratch can be found in the 5G EVE Github repository[30].

3.4.2 Open API description

The updated OpenAPI specification of the RC Handler is available in the 5G EVE Github repository[31], and are also presented in Table 4. It is mostly the same than the one reported in deliverable D3.4 but including the nsInstanceId in the request to provide the configuration for the Infrastructure Day-2 configuration. The description of the OpenAPI is presented in Table 5.

Table 5: Runtime Configurator operations from its Open API specification.

Service	Path	Method	Input	Output[32]	Description
RC Handler	/rc/nbi	GET	-	200 - OK	Get RC version and check that the service is running.

[30] Deployment steps available here: https://github.com/5GEVE/5geve-wp3-rc-deployment (tag v0.3).

[31] Specification available here: https://github.com/5GEVE/OpenAPI/tree/master/RuntimeConfigurator (v3 file).

[32] The output is always 200 OK, but the operation status is managed thanks to the operation status attribute which is always returned in all requests.

RC Handler	/rc/nbi/application/day2/configuration	POST	configurationScript = scripts executed during start application Day-2 configuration operation. resetConfigScript = scripts executed during reset application Day-2 configuration operation.	200 - OK	Provide the scripts for application Day-2 configuration to the RC. It returns the configId and the current status of the operation.
RC Handler	/rc/nbi/application/day2/configuration/<configId>	GET	configId	200 - OK	It returns the current status of the operation.
RC Handler	/rc/nbi/application/day2/configuration/<configId>/start	POST	configId	200 - OK	Execute the configurationScript commands. It returns the current status of the operation.
RC Handler	/rc/nbi/application/day2/configuration/<configId>/reset	POST	configId	200 - OK	Execute the resetConfigScript commands. It returns the current status of the operation.
RC Handler	/rc/nbi/application/day2/configuration/<configId>/abort	DELETE	configId	200 - OK	Abort the current operation. It returns the current status of the operation.
RC Handler	/rc/nbi/infrastructure/day2/configuration	POST	List<infrastructureMetricsInfo> = list containing the data related to each infrastructure metric to be configured. (Optional) nsInstanceId = network service related to these infrastructure metrics.	200 - OK	Provide the information related to each infrastructure metric for the Day-2 configuration process. It returns the configId and the current status of the operation.
RC Handler	/rc/nbi/infrastructure/day2/configuration/<configId>	GET	configId	200 - OK	It returns the current status of the operation.

RC Handler	/rc/nbi/infrastructure/day2/configuration/<configId>/start	POST	configId	200 - OK	Execute the configurationScript commands (extracted from the IWF Repository by the RC). It returns the current status of the operation.
RC Handler	/rc/nbi/infrastructure/day2/configuration/<configId>/stop	POST	configId	200 - OK	Execute the stopConfigScript commands (extracted from the IWF Repository by the RC). It returns the current status of the operation.
RC Handler	/rc/nbi/execution	POST	execScript = commands to be executed during Experiment Execution phase. executionId = identification provided by the EEM.	200 - OK	Provide the scripts for experiment execution to the RC. It return the execId and the current status of the operation.
RC Handler	/rc/nbi/execution/<execId>	GET	execId	200 - OK	It returns the current status of the operation.
RC Handler	/rc/nbi/execution/<execId>/start	POST	execId	200 - OK	Execute the execScript commands. It returns the current status of the operation.
RC Handler	/rc/nbi/execution/<execId>/abort	DELETE	execId	200 - OK	Abort the current operation. It returns the current status of the operation.

3.4.3 Service description

Compared to the documentation provided in deliverable D3.4, there have been significant changes in the information model used to build the scripts to be placed in both the Test Cases Blueprints (TCBs) and the IWF Repository. In this way, a generic model has been proposed to hide the complexity of invoking Ansible playbooks to the verticals, moving this complexity to the RC Handler. Apart from that, the specific-purpose workflows related to the RC will be also updated with the integration of the interaction between the RC and the MSNO.

3.4.3.1 Final information model to build the scripts

The new version of the RC reported in this deliverable defines an information model for the scripts provided by the users in both the TCBs and IWF Repository. The operations that can be used are the following:

- **Execute command[33]:** this operation involves the execution of the command provided as parameter. The format of this operation is the following: *EXECUTE_COMMAND[_WINDOWS] <IP_SERVER> <USER>:<PASSWORD> <COMMAND>*, where:
    - *EXECUTE_COMMAND[_WINDOWS]*: reserved word to make reference to this operation. If the command is executed in a Linux server, just include *EXECUTE_COMMAND* here. However, if the targeted server is a Windows server, just include *EXECUTE_COMMAND_WINDOWS* here.
    - *<IP_SERVER>*: IP address of the VNF/PNF/infrastructure in which the instruction will be executed. In case of using this operation for Application Day-2 configuration or Execution requests (defined in the TCBs), you have to include the reference to this parameter in the *infrastructureParameters* field. However, if it is related to Infrastructure Day-2 configuration (defined in the IWF Repository), you have to include *\$\$ipAddress* here.
    - *<USER>:<PASSWORD>*: user and password to access the VNF/PNF/infrastructure. In case of using this operation for Application Day-2 configuration or Execution requests (defined in the TCBs), you have to include the reference to this parameter in the *userParameters* field. However, if it is related to Infrastructure Day-2 configuration (defined in the IWF Repository), you have to include *\$\$username:\$\$password* here.
    - *<COMMAND>*: directly include the command to be executed. Remember that, if you need to pass parameters to the command (for scripts in the TCBs), you have to include them as *userParameters*.
- **Sleep[34]:** this operation just executes a sleep. It is useful for steps in which manual interaction with the components related to the experiment have to be managed, or just to do nothing. The way of invoking this operation is: *SLEEP <SLEEP_TIME>*, where:
    - *SLEEP*: reserved word to make reference to this operation.
    - *<SLEEP_TIME>*: time to be slept. This parameter has to be defined in the *userParameters* field in the TCBs, or including it manually in case of being related to Infrastructure Day-2 configuration (i.e. it is saved in the IWF Repository).
- **Install Filebeat[35]:** this operation enables the installation of Filebeat as Data Shipper in the targeted server, configuring it to start monitoring a specific metric. The format to be followed for this operation is the following: *INSTALL_FILEBEAT <IP_SERVER> <USER>:<PASSWORD> <METRIC_ID> <TOPIC_NAME> <SITE> <UNIT> <INTERVAL> <DEVICE_ID> <MONITORED_FILE_PATH>*, where:
    - *INSTALL_FILEBEAT:* reserved word to make reference to this operation.
    - *<IP_SERVER>*: IP address of the VNF/PNF/infrastructure in which the instruction will be executed. In case of using this operation for Application Day-2 configuration or Execution requests (defined in the TCBs), you have to include the reference to this parameter in the *infrastructureParameters* field. However, if it is related to Infrastructure Day-2 configuration (defined in the IWF Repository), you have to include *\$\$ipAddress* here.
    - *<USER>:<PASSWORD>*: user and password to access the VNF/PNF/infrastructure. In case of using this operation for Application Day-2 configuration or Execution requests (defined in the TCBs), you have to include the reference to this parameter in the *userParameters* field. However, if it is related to Infrastructure Day-2 configuration (defined in the IWF Repository), you have to include *\$\$username:\$\$password* here.
    - *<METRIC_ID>*: reference to the metricId. In case of using this operation for Application Day-2 configuration (defined in the TCBs), you have to include the reference to this parameter in

[33] An example of this operation defined in TCBs can be found here: https://github.com/5GEVE/blueprint-yaml/blob/master/UC_0.1_ApacheSimple/Support_tools/TCB/Simple_UC_TCB.json.

[34] An example of this operation defined in TCBs can be found here: https://github.com/5GEVE/blueprint-yaml/blob/a22b152ccd3808d5f771c3d6e92a027b1a41a92a/UC_ict19_5GSolutions/TCB_5G_Solutions.json.

[35] Instructions to use this operation can be found here: https://github.com/5GEVE/5geve-rc/tree/master/install_filebeat.

the *infrastructureParameters* field. However, if it is related to Infrastructure Day-2 configuration (defined in the IWF Repository), you have to include *\$\$metric_id* here.

- <*TOPIC_NAME*>: reference to the topic name. In case of using this operation for Application Day-2 configuration (defined in the TCBs), you have to include the reference to this parameter in the *infrastructureParameters* field. However, if it is related to Infrastructure Day-2 configuration (defined in the IWF Repository), you have to include *\$\$topic_name* here.
- <*SITE*>: reference to the site facility, to use the proper Kafka Site Broker IP address. In case of using this operation for Application Day-2 configuration (defined in the TCBs), you have to include this in the following way*: __*<*SITE_NAME*> (e.g. *__SPAIN_5TONIC*). However, if it is related to Infrastructure Day-2 configuration (defined in the IWF Repository), you have to include *\$\$broker_ip_address* here.
- <*UNIT*>: reference to the unit of the metric. In case of using this operation for Application Day-2 configuration (defined in the TCBs), you have to include the reference to this parameter in the *infrastructureParameters* field. However, if it is related to Infrastructure Day-2 configuration (defined in the IWF Repository), you have to include *\$\$unit* here.
- <*INTERVAL*>: reference to the interval to gather the metric. In case of using this operation for Application Day-2 configuration (defined in the TCBs), you have to include the reference to this parameter in the *infrastructureParameters* field. However, if it is related to Infrastructure Day-2 configuration (defined in the IWF Repository), you have to include *\$\$interval* here.
- <*DEVICE_ID*>: reference to the deviceId. In case of using this operation for Application Day-2 configuration (defined in the TCBs), you have to include the reference to this parameter in the *infrastructureParameters* field. However, if it is related to Infrastructure Day-2 configuration (defined in the IWF Repository), you have to include *\$\$deviceId* here. If this parameter is not used in any case, just include *nil*.
- <*MONITORED_FILE_PATH* >: reference to the file path to be monitored by Filebeat. In case of using this operation for Application Day-2 configuration (defined in the TCBs), you have to include the reference to this parameter in the *userParameters* field. However, if it is related to Infrastructure Day-2 configuration (defined in the IWF Repository), you have to include the explicit file path. The format of this file path must be the following: */var/log/<METRIC_ID>.log.*

For this last operation related to Filebeat, note that the configuration provided in the targeted server is the installation of Filebeat, listening to changes in the */var/log/<METRIC_ID>.log* to publish the data included there to the Monitoring system. However, the way of feeding that log file is not included and must be provided in a separate command. To do this, note the following considerations:

- The script to feed the log file can make use of a configuration file provided by the Filebeat operation, which is placed in */usr/bin/<METRIC_ID>-day2-config.yml*, which is a YAML file with the following content:

```
broker_ip_address: <Kafka broker IP address in the given site facility – static parameter>
topic_name: <topic to be used for publishing in Kafka – generated in the ELM>
device_id: <ID of the device – from NSO/customized in the TCB>
unit: <unit parameter - from blueprints>
interval: <interval parameter - from blueprints>
```

- The way that monitoring data must be saved in the log file is by following the CSV format proposed in deliverable D3.4, section 3.4.3.3, related to the information model to publish monitored data in the Monitoring platform. It is the following (*device_id* and *context* are optional fields, which must be set to 'nil' if they are not used. In the case of the *context* field, if used, it must contain the values in the following format: *param1=value1 param2=value2*, etc., i.e. param=value items separated by spaces).

```
<metric_value>,<timestamp>,<unit>,[<device_id>],[<context>]
```

In this case, however, the batch option is not possible. Note that, again, *device_id* and *context* are optional fields, which must be set to 'nil' if they are not used.

An example of a TCB making use of all these three operations is provided below. The following TCB is based on two VNFs, configuring one metric on each VNF: *track_device* in VNFA (using Filebeat) and *delay_iface* in VNFB (using a specific script[36]). Note that, if more than one script is provided in the *configurationScript/executionScript/resetConfigScript* fields, they have to be separated by a ";". The example is the following:

```
testCaseBlueprint:
  description: Example for integration with the RC
  name: Example TCB
  configurationScript: INSTALL_FILEBEAT vnf.<vnfdA_id>.extcp.<extcp_id>.ipaddress
$$userA:$$passwordA track_device $metric.topic.track_device $$metric.site.track_device
$$metric.unit.track_device $$metric.interval.track_device $$metric.deviceId.track_device
$$monitoredPath1;
  EXECUTE_COMMAND vnf.<vnfdB_id>.extcp.<extcp_id>.ipaddress $$userB:$$passwordB "/bin/bash
/home/eve/install_datashipper.sh delay_iface $$metric.topic.delay_iface $$metric.site.delay_iface
$$metric.unit.delay_iface $$metric.interval.delay_iface $$metric.deviceId.delay_iface";
  EXECUTE_COMMAND vnf.<vnfdA_id>.extcp.<extcp_id>.ipaddress $$userA:$$passwordA "sudo apt-get
install python3"
  executionScript: EXECUTE_COMMAND vnf.<vnfdA_id>.extcp.<extcp_id>.ipaddress $$userA:$$passwordA
"ping -c5 vnf.<vnfdB_id>.extcp.<extcp_id>.ipaddress";
  SLEEP $$sleepTime;
  EXECUTE_COMMAND vnf.<vnfdB_id>.extcp.<extcp_id>.ipaddress $$userB:$$passwordB "ping -c5
vnf.<vnfdA_id>.extcp.<extcp_id>.ipaddress";
  resetConfigScript: EXECUTE_COMMAND vnf.<vnfdA_id>.extcp.<extcp_id>.ipaddress $$userA:$$passwordA
"sudo systemctl stop filebeat";
  EXECUTE_COMMAND vnf.<vnfdB_id>.extcp.<extcp_id>.ipaddress $$userB:$$passwordB "/bin/bash
/home/eve/stop_datashipper.sh";
  userParameters:
    userA: $$userA
    passwordA: $$passwordA
    userB: $$userB
    passwordB: $$passwordB
    monitoredPath1: $$monitoredPath1
    sleep_time: $$sleepTime
  infrastructureParameters:
    $$metric.topic.track_device
    $$metric.site.track_device
    $$metric.unit.track_device
    $$metric.interval.track_device
    $$metric.deviceId.track_device
    $$metric.topic.delay_iface
    $$metric.site.delay_iface
    $$metric.unit.delay_iface
    $$metric.interval.delay_iface
    $$metric.deviceId.delay_iface
    vnf.<vnfdA_id>.extcp.<extcp_id>.ipaddress
    vnf.<vnfdB_id>.extcp.<extcp_id>.ipaddress
  version: '2.0'
```

Although a specific example for each operation and a general example using all operations have been provided, note that there are examples of Test Case Blueprints that can be used as base to build TCBs for other use cases in the 5G EVE Github repository[37].

[36] In fact, this example can be used as base to invoke commands that triggers scripts in the targeted servers that provides the data needed by the Data Shippers to publish the data in the corresponding Site Broker.

[37] Blueprints available here: https://github.com/5GEVE/blueprint-yaml. Just access to the different use cases and check the TCBs provided.

Moreover, for the definition of Data Shippers in the IWF Repository, the following example is provided as a reference. These scripts have to be provided to the partner responsible for the IWF Repository for including them in the repository:

```
  "dataShipperId": "SPAIN_5TONIC.LATENCY_USERPLANE",
  "ipAddress": "1.2.3.4",
  "username": "user",
  "password": "password",
  "configurationScript": “EXECUTE_COMMAND \$\$ipAddress \$\$username:\$\$password \"/home/user
/add_metric.sh \$\$metric_id \$\$broker_ip_address \$\$topic_name \$\$device_id \$\$unit
\$\$interval\"; EXECUTE_COMMAND \$\$ipAddress \$\$username:\$\$password \"/home/user/start.sh
\$\$metric_id\";",
  "stopConfigScript": “EXECUTE_COMMAND \$\$ipAddress \$\$username:\$\$password
\"/home/user/stop.sh \$\$metric_id\";",
  "metricType": "CPU_CONSUMPTION"
```

Some particularities must be considered when creating a script to be placed in the IWF Repository:

- The dataShipperId must be composed in the following way: <*SITE*>. <*METRIC_ID*>.
- The *configurationScript* and *stopConfigScript* fields must include the references to the fields related to the infrastructure metric (e.g., topic name, unit, interval, etc.) in order to do the translation in the RC.
    - If your Data Shipper is using VNF IP addresses obtained from the MSNO, the reference for this field is */$/$vnfIpAddresses*. The IPs are provided in a comma-separated string (e.g., 1.2.3.4,5.6.7.8).
- The metricType is related to the *iMetricType* defined in the blueprints.

3.4.3.2 Final specific-purpose workflows

The workflows related to the operation of the RC are exactly the same to the ones presented in deliverable D3.4, but just changing the step number 3, related to the starting of the infrastructure Day-2 configuration process, as the interaction with the MSNO may be needed. The workflow including this interaction is presented in Figure 23, which only includes an optional interaction with the MSNO prior to the interaction with the IWF Repository, only triggered if the *nsInstanceId* is provided in the request from the EEM.

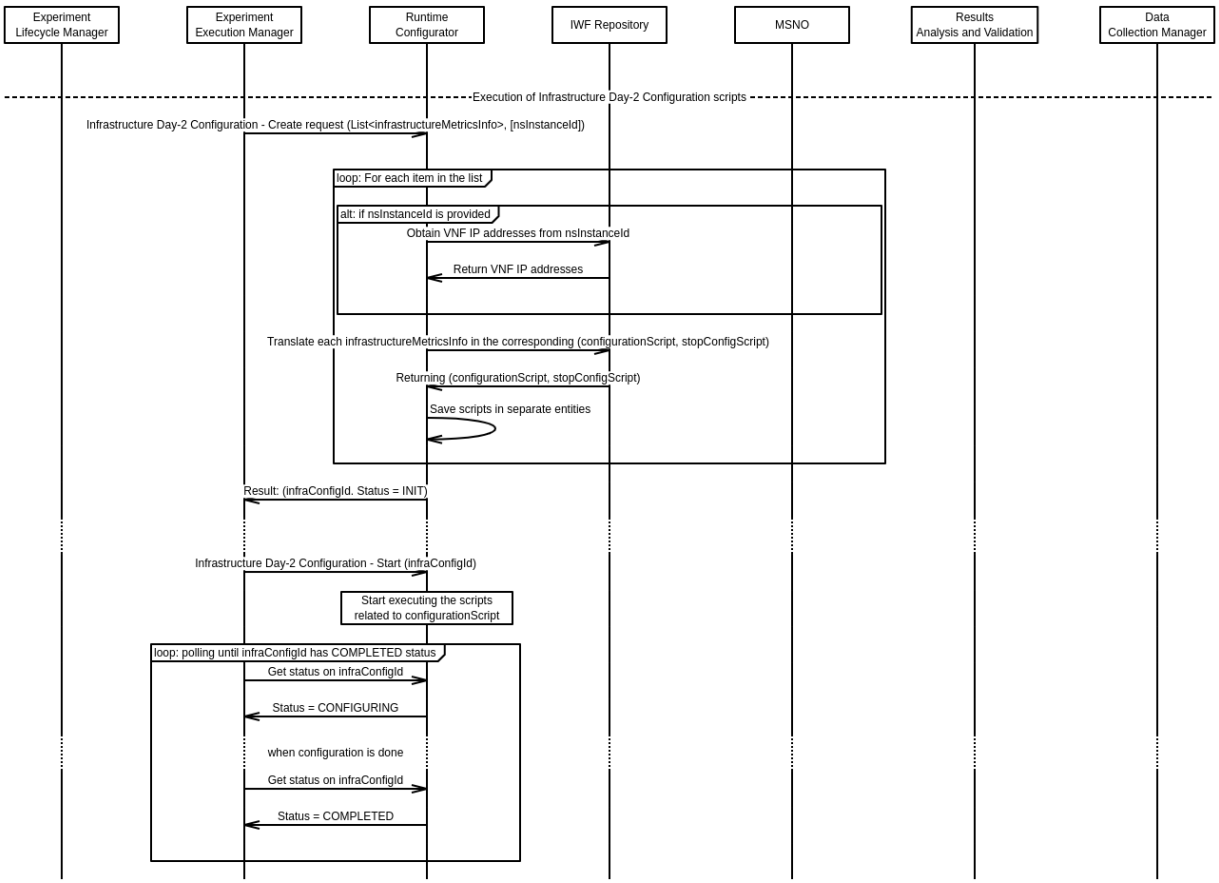

Figure 23: Execution of infrastructure Day-2 configuration scripts

3.5 Adaptation Layer

As already commented in the deliverable D3.4, the Adaptation Layer at the South-Bound Interface (SBI) provides a common access interface to trial site services and resources. Specifically, multiple software components build the overall implementation, and they are distinct based on the specific feature to be accessed (i.e., orchestration, configuration, monitoring). In D3.2 ([2]) we report a complete presentation of requirements, features, and high-level design. The following subsections provide a description of the implementation process, software architecture and technologies for each part of the Adaptation Layer.

3.5.1 Multi-Site Catalogue SBI

The Multi-Site Catalogue described in Section 3.1 includes SBI operations for the management of NSDs, VNFDs and PNFDs in the local NFVOs. Furthermore, it features a synchronization mechanism to keep the Interworking Layer aligned with the information available in the local NFVO catalogues. To achieve this, the Multi-Site Catalogue is structured with a driver-based architecture. Each driver is in charge of the translation of the NSD, VNFD, and PNFD from the standard ETSI NFV SOL 001 [8] model to the specific model of the targeted NFVO type. Reverse translation is supported as well. The driver also implements the required adaptations to interact with the NSD Management API of the specific NFVO.

In this software release of the Multi-Site Catalogue includes an OSM driver (that is compatible with OSM R4, R5, R6, R7 and R8) that allows to integrate with the Italian, Spanish and Greek site facilities (i.e., all of them make use of OSM as site orchestrator), a driver to integrate the ONAP NFV Orchestrators and a third driver named 5Growth. More details can be found in Section 3.1.1.

3.5.2 Multi-Site NSO to local Orchestrator's interface

As already commented in the deliverable D3.4, the Multi-Site NSO to local Orchestrator interface (MSO-LO) is the portion of the Adaptation Layer that enables the Interworking Layer to access the orchestration features provided by the trial sites involved in the 5G EVE project. Trial sites can expose one or more local Network Function Virtualization Orchestrator (NFVO), each one managing a different set of resources.

This interface provides the following features:

1. Operations to retrieve information about local NFVOs registered with the 5G EVE platform.
2. Operations to retrieve, create, instantiate, scale, terminate, and delete NS instances.
3. Operations to retrieve, create, and delete subscriptions to notifications about the status of one or more NS instance.

3.5.2.1 Application architecture

The application architecture already developed in the previous software release continues to be applicable. For the sake of completeness, a brief description of its main components is provided below. MSO-LO is a web service that exposes an NBI based on the REST architectural style. To implement this module, we mainly used tools for API description / documentation and tools for web service implementation. Due to the technology used, a reverse proxy is used to bundle all communications between MSO and the MSO-LO backend.

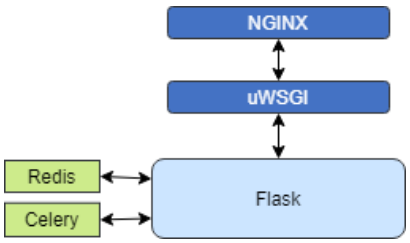

Figure 24: Architecture of the application

As shown in Figure 24, the architecture of the application is composed by the following components:

- Reverse proxy: we use the popular NGINX[38].
- WSGI server: we use uWSGI[39] typically used for running Python web applications.
- Webservice: we use the lightweight micro-framework Flask[40] written in python, that is designed for a quick and easy "getting started", with the ability to scale up to complex applications.
- In-memory data store: we use Redis[41], an open source (BSD licensed), in-memory data structure store, used as a database, cache and message broker.
- Periodic and background tasks manager: we use Celery[42] , an open source asynchronous task queue or job queue which is based on distributed message passing.

Then the execution flow is as follows: a generic HTTP Client initiates a request to NGINX which forwards the request to uWSGI using a Unix socket. Then uWSGI forwards the request to Flask which handles the request and generates the response, finally the response is returned across the stack.

[38] https://www.nginx.com/

[39] https://uwsgi-docs.readthedocs.io/en/latest/

[40] https://flask.palletsprojects.com/en/1.1.x/

[41] https://redis.io/

[42] https://docs.celeryproject.org/en/stable/

Concerning the persistent storage, we use the IWF Repository component. Furthermore, we use a Redis instance as an in-memory storage database to support the background task scheduler (celery), the tracking of LCM operation status and shared information between multiple processes of the MSO-LO.

3.5.2.2 Software architecture

With respect to the previous release the working principle of the software architecture remains the same, but new driver implementations are added, specifically we support two new drivers EVER, which supports the new Radio Access Network Orchestrators, RANO, (EVER, Network Controller) and FIVEGR_SO, which is used to integrate 5Growth (ICT-19 project) sites into 5G EVE framework. As mentioned in D3.4 the goal of MSO-LO is providing a common interface to access the features of various NFVO types without restrictions on open-source or proprietary licenses. In order to ensure the dual objective of supporting the NFVO implementations currently declared in the 5G EVE project and supporting the inclusion of new NFVO technologies in the future, we have designed the software with a driver-based architecture as illustrated in Figure 25.

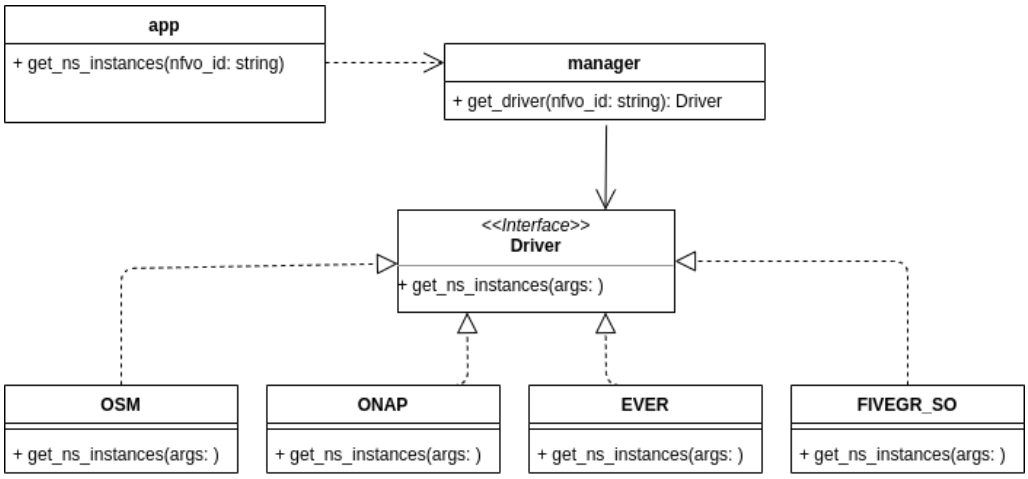

Figure 25: UML class diagram with an example method of the interface.

Figure 25 shows a simplified version of the UML class diagram with just one method of the API, which is enough to explain how the software works. As we use Python for the software implementation, not all the classes shown in the diagram are Python classes. Some entities are just modules, as they do not need to store any status but are a mere collection of static methods.

The *app* module contains the declaration of the REST API paths and related HTTP operations. Furthermore, it includes the mapping between the previous entities and the relevant Python method. The method implementation here is generic and agnostic from the specific type of local NFVO.

The other entities in the figure implement the factory method design pattern [13]. The pattern is very effective for our solution as it separates the creation of driver instances from the *app* module that actually uses them. Indeed, the *app* module depends (dashed arrow) from the *manager* module for the selection of the specific NFVO implementation. The *manager* module implements the factory method plus some utility methods for the interaction with the NFVO database.

The *Driver* abstract class realizes a contract for the driver implementation and is implemented by means of the Abstract Base Class Python module (ABC[43]). The addition of a specific NFVO driver simply consists in the extension of *Driver* and in the implementation of all its declared methods.

[43] https://docs.python.org/3/library/abc.html

To better understand the interactions between the modules and class instances, we use a message sequence chart. Figure 26 shows the interactions needed to obtain the list of NS instances from the catalogue of an OSM NFVO identified with 'osm_1'.

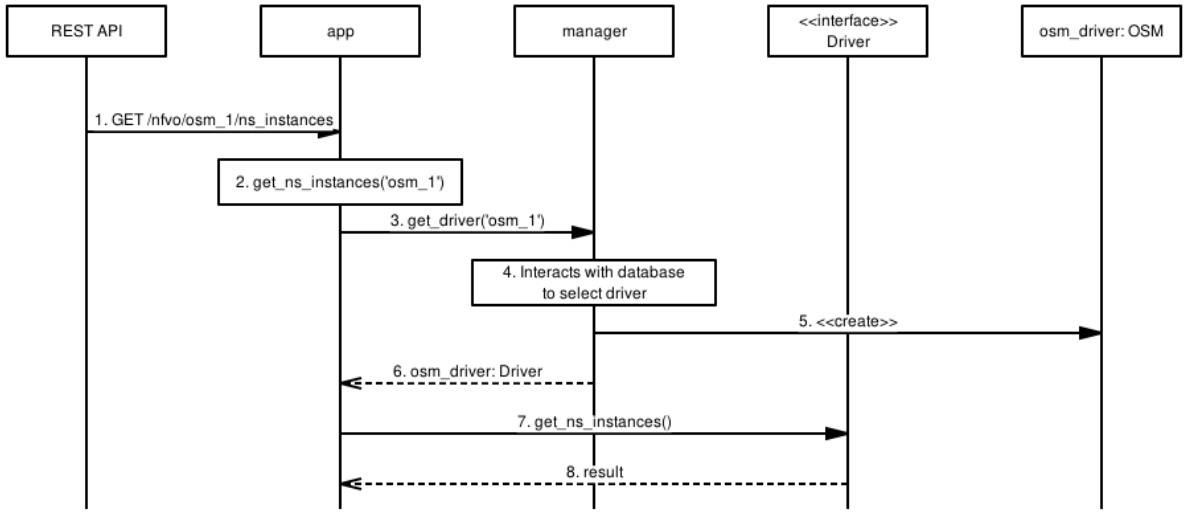

Figure 26: Message sequence chart to show modules and objects interactions.

At step 1, a GET request to the path "/nfvo/osm_1/ns_instances" is routed to the *app* module. The *app* module executes the relevant method (step 2). At step 3, the *app* module requests the appropriate driver to the *manager*. The *manager* determines the appropriate driver (step 4), creates an instance of the OSM driver (step 5), and returns it to the *app* module (step 6). The *app* module actually receives an object of generic type *Driver*. In step 7, *app* calls the relevant method and in step 8 it receives the result.

Regarding the NS Lifecycle Management Notification, the MSO-LO provides an API interface compliant to the ETSI NFV SOL005 specification, supporting three types of notifications:

- NsLcmOperationOccurrenceNotification: Inform subscribed clients about an event in the lifecycle of an NS. The notification includes NS id, OpOcc id, StateType (PROCESSING, COMPLETED, etc.) and affected components.
- NsIdentifierCreationNotification: Informs subscribed clients about NS id creation.
- NsIdentifierDeletionNotification: Informs subscribed clients about NS id deletion.

The solution provides a generic subscription system to allow clients to filter out notifications and receive only the ones about NS instances they are interested about. Subscriptions are stored in the IWF Repository and the MSO-LO northbound interface provides CRUD methods for their management. More details on supported REST operations can be found in Section 3.5.2.4. A crucial part of the subscription information element is the "uri". This element must be a valid HTTP endpoint enabled by the subscribing client to receive notifications. MSO-LO sends notifications via a POST request targeting the aforementioned "uri".

The notifications management is delegated to specific components, responsible for checking the status of operations applied to NS instances for each type of NFVO. To better understand how the specific components works, we use a message sequence chart for each of them to show the interaction of the MSO-LO with the other components of the 5G EVE ecosystem.

Figure 27 shows the case of the ONAP NFVO located in the French Site. At first step, the MSNO sends a request to instantiate an NS Instance, and the MSO-LO takes care to handle the request forwarding it to the specific ONAP Translation Component. At step 2 the MSNO sends a request in order to create a subscription for a specific NS Instance to the MSO-LO (the one that has been just created) and, if the request is valid, the subscription is stored inside the IWF Repository.

As soon as the ONAP Translation Component has a change of state for an operation to be notified, it sends a POST on the SBI of the MSO-LO, specifically on the endpoint "/nfvo/{nfvoId}/notifications". At this point,

the MSO-LO takes care of the notification and checks whether there are any subscriptions related to the NS Instance involved and it forwards the notification on the URI that was defined by the MSNO during the creation of the subscription.

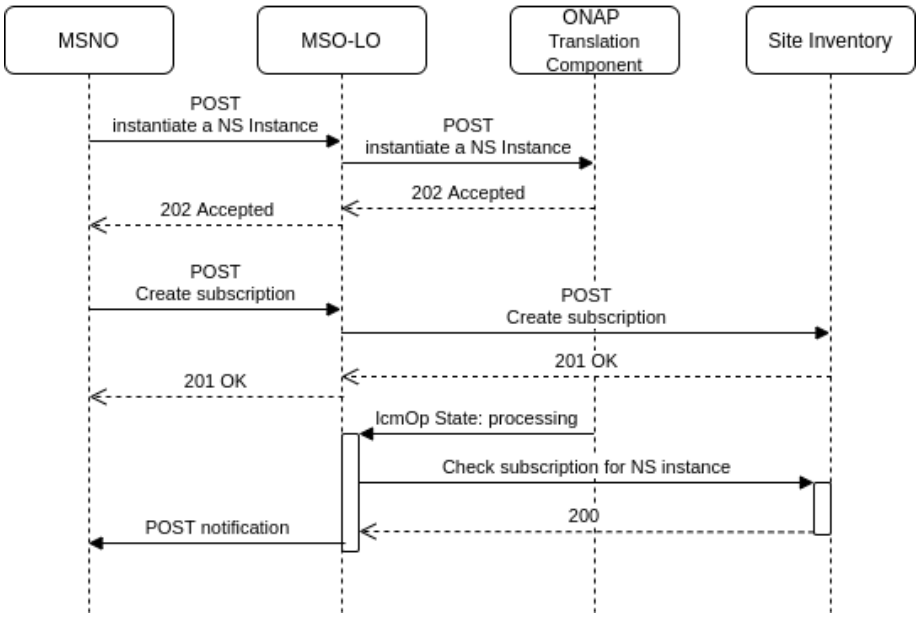

Figure 27: Subscriptions and notifications management for ONAP

Figure 28 shows the case of the OSM NFVO. The first two steps, the instance creation and the subscription creation, are similar to the use case of the ONAP NFVO described above. Contrary to the previous case, the OSM NFVO is not able to independently notify a change of state happening on one of its managed resources. Then, a background process is activated within the MSO-LO that periodically check for any operations happening in the OSM NFVO. The last state of a specific operation is stored in an in-memory storage implemented with Redis, a key-value database. If a change in the operation's state is detected, the MSO-LO takes care of forwarding the notification to the subscribed client.

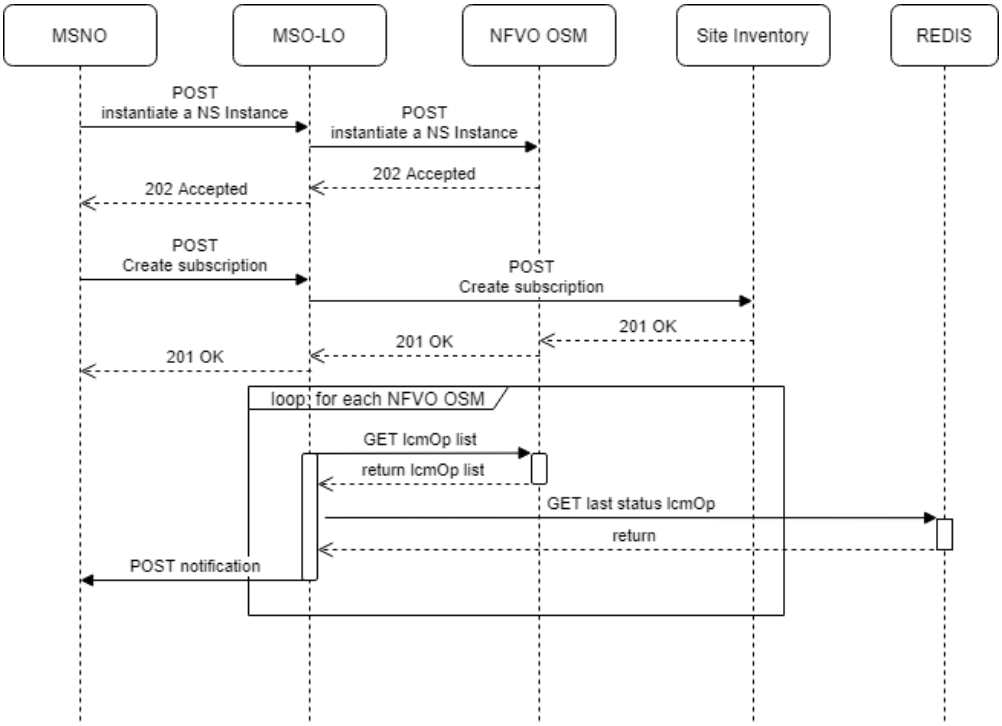

Figure 28: Subscriptions and notifications management for OSM

3.5.2.3 OSM specific features

The OSM specific features remains the same as described in D3.4, in particular the sites hosting the OSM NFVO have shown the need to support additional parameters during the instantiation phase. To meet this request the MSO-LO, in accordance with the ETSI NFV-SOL 005 ([5]) standard, supports the "additionalParamsForNs" field in the body of requests to the endpoint "/nfvo/{nfvoId}/ns_instance/{nsInstanceId}/instantiate". The information included in this field is only processed by the OSM driver, while other drivers ignore it.

The following instantiation parameters are supported:

- vim network name: this allows the association of a VLD to an existing network in the targeted site, useful when there is a static management network.
- vim account: this allows the MSO to specify in which VIM a particular VNF must be instantiated, useful in case there are multiple VIMs within a site.

3.5.2.4 Open API description

It is possible to view the updated version of the API specification in the 5G EVE public organization hosted on GitHub[44]. Table 6 and Table 7 summarize in tabular format the contents of the YAML file containing the API methods offered by the MSO-LO interface, respectively for the NFVO and RANO interfaces.

Table 6: MSO-LO API interface for NFVO

Service	Path	Method	Input	Output	Description
MSO-LO	/nfvo	GET	-	Array of NFVO info	Retrieve list of local NFVO.

[44] https://github.com/5GEVE/OpenAPI

MSO-LO	/nfvo/{nfvoId}	GET	-	NFVO info	Read an individual NFVO.
MSO-LO	/nfvo/{nfvoId}/ns_instances	POST	nsdId, nsName, nsDescription	NS identifier	Creates and returns a Network Service identifier (nsId) in a Nfvo
MSO-LO	/nfvo/{nfvoId}/ns_instances	GET	-	Array of NS instances	Retrieve list of NS instances.
MSO-LO	/nfvo/{nfvoId}/ns_instances/{nsInstanceId}	GET	-	NS instance	Read an individual NS instance resource.
MSO-LO	/nfvo/{nfvoId}/ns_instances/{nsInstanceId}	DELETE	-	-	Delete an individual NS instance resource.
MSO-LO	/nfvo/{nfvoId}/ns_instance/{nsInstanceId}/instantiate	POST	nsFlavourId	-	Instantiate a NS instance.
MSO-LO	/nfvo/{nfvoId}/ns_instance/{nsInstanceId}/terminate	POST	terminationTime	-	Terminate a NS instance.
MSO-LO	/nfvo/{nfvoId}/subscriptions	POST	filter, callbackUri	subscription identifier	Subscribe to NS lifecycle change notifications.
MSO-LO	/nfvo/{nfvoId}/ns_lcm_op_occs	GET	-	Array of NS LCM operation	Query multiple NS LCM operation
MSO-LO	/nfvo/{nfvoId}/ns_lcm_op_occs/{nsLcmOpOccId}	GET	nsLcmOpOccId	NS LCM operation	Read an individual NS LCM operation occurrence resource.
MSO-LO	/nfvo/{nfvoId}/subscriptions	GET	-	Array of subscription information	Query multiple subscriptions.
MSO-LO	/nfvo/{nfvoId}/subscriptions/{subscriptionId}	GET	-	subscription information	Read an individual subscription resource.
MSO-LO	/nfvo/{nfvoId}/subscriptions/{subscriptionId}	DELETE	-	-	Terminate a subscription.
MSO-LO	/nfvo/{nfvoId}/notification	POST	nsInstanceId operation	-	Notify a status change of a

			operationState		operation state to MSO-LO

Table 7: MSO-LO API interface for RANO

Service	Path	Method	Input	Output	Description
MSO-LO	/rano	GET	-	Array of RANO info	Retrieve list of local RANO.
MSO-LO	/rano/{ranoId}	GET	-	RANO info	Read an individual RANO.
MSO-LO	/rano/{ranoId}/ns_instances	POST	nsdId, nsName, nsDescription	NS identifier	Creates and returns a Network Service identifier (nsId) in a Nfvo
MSO-LO	/rano/{ranoId}/ns_instances	GET	-	Array of NS instances	Retrieve list of NS instances.
MSO-LO	/rano/{ranoId}/ns_instances/{nsInstanceId}	GET	-	NS instance	Read an individual NS instance resource.
MSO-LO	/rano/{ranoId}/ns_instances/{nsInstanceId}	DELETE	-	-	Delete an individual NS instance resource.
MSO-LO	/rano/{ranoId}/ns_instance/{nsInstanceId}/instantiate	POST	nsFlavourId	-	Instantiate a NS instance.
MSO-LO	/rano/{ranoId}/ns_instance/{nsInstanceId}/terminate	POST	terminationTime	-	Terminate a NS instance.
MSO-LO	/rano/{ranoId}/ns_lcm_op_occs	GET	-	Array of NS LCM operation	Query multiple NS LCM operation
MSO-LO	/rano/{ranoId}//ns_lcm_op_occs/{nsLcmOpOccId}	GET	nsLcmOpOccId	NS LCM operation	Read an individual NS LCM operation occurrence resource.

3.6 IWF Repository

As stated in deliverable D3.4 the 5G EVE IWF Repository is the component in charge of storing shared information about sites and their features. It acts as a centralized storage element, exposing its data to other components of the Interworking Framework via a REST HTTP interface. The source code can be found in GitHub[45].

3.6.1 Software architecture

With respect to the previous release the working principle of the software architecture remains the same, but in this release the major changes concern the information elements, in particular Figure 30 shows an updated and detailed Entity-Relationship diagram of the generated database. The IWF Repository is a REST HTTP Java application using a PostgreSQL database for persistent storage. To enforce best practices and speed up the development process, we use the Spring Framework[46] and the Spring Boot configuration convention. Figure 29 shows the IWF components that make use of the IWF Repository in their operational workflows.

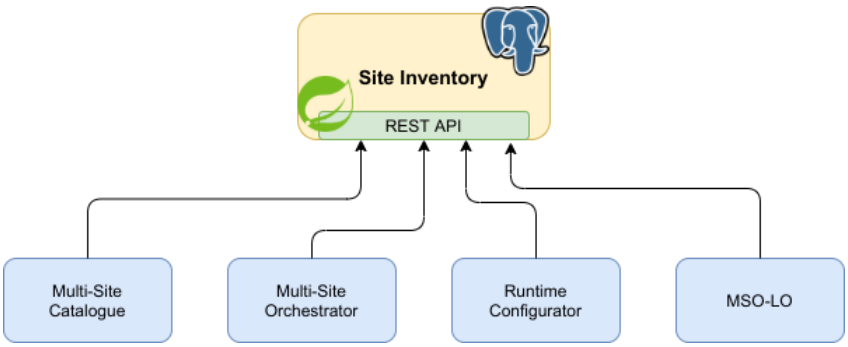

Figure 29: IWL components that use the IWF Repository.

Information elements as mentioned in D3.4 are encoded as Java classes and annotated with '@Entity' following the standard defined by the Java Persistence API. The Spring Framework, thanks to Hibernate[47], uses the annotations to define the data structures in PostgreSQL using the proper Data Definition Language. No direct interaction with the database is required from the developer. The approach made it possible to create a complex database schema in a short time.

[45] https://github.com/5GEVE/iwf-repository

[46] https://spring.io/

[47] https://hibernate.org/

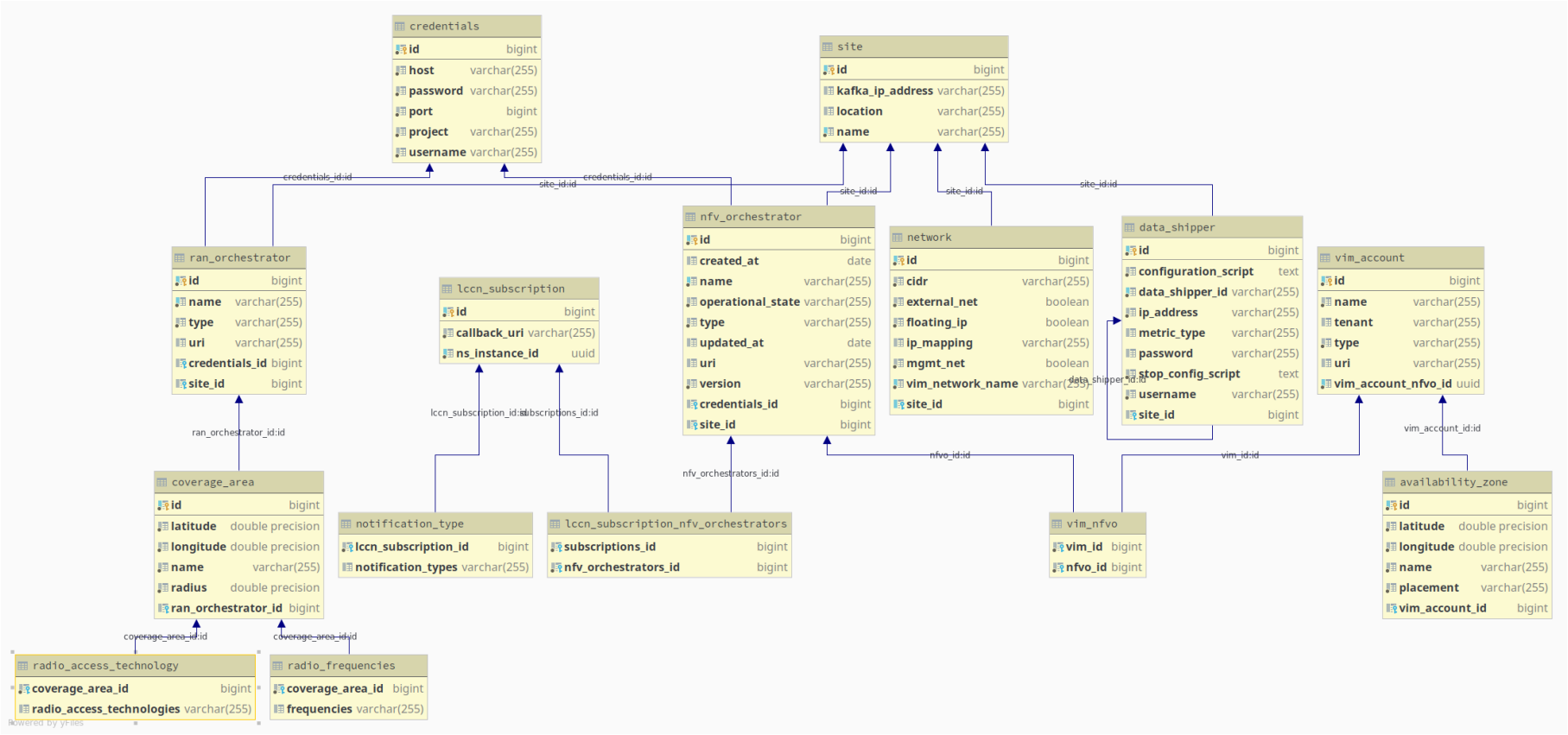

Figure 30: Entity-Relationship diagram for IWF Repository database

To expose the entities and related operation on an HTTP REST interface, we use the module 'spring-data-rest'. This Spring module automatically exposes a discoverable REST APIs by discovering the entities defined in the project. The API supports CRUD operations in HTTP format, such as POST, GET, PATCH/PUT, DELETE and generates sub-paths to navigate the associations. The amount of code and development effort to generate all this is extremely limited. As for the previous release, we use Dockerfile + docker-compose. The Dockerfile downloads all necessary dependencies and builds the application. The docker-compose creates a multi-service environment including both the IWF Repository application and a PostgreSQL database instance. The application is extremely portable, and it can be deployed on any environment with zero configuration.

3.6.2 Open API description

It is possible to view the updated OpenAPI documentation for IWF Repository interface in the 5G EVE public organization hosted on GitHub[48]. Since it is auto generated, the documentation is very verbose.

Table 8: IWF Repository main REST API paths

Service	Path	Method	Input	Output	Description
site-inventory	/availabilityZones	GET	-	Array of Availability Zone entities	Retrieve list of Availability Zones entities.
site-inventory	/availabilityZones	POST	Availability Zone entity	The newly created entity	Create new Availability Zone entity.
site-inventory	/availabilityZones/{id}	GET	-	Availability Zone entity	Retrieve a single Availability Zone entity
site-inventory	/availabilityZones/{id}	PUT	Availability Zone entity with updated values	Availability Zone entity	Update an existing Availability Zone entity.
site-inventory	/availabilityZones/{id}	DELETE	-	-	Delete an existing Availability Zone entity.
site-inventory	/availabilityZones/{id}	PATCH	Fields to be updated	Availability Zone entity	Update an existing Availability Zone entity.
site-inventory	/dataShippers	GET	-	Array of Data Shipper entities	Retrieve list of Data Shipper entities.
site-inventory	/dataShippers	POST	Data Shipper entity	The newly created entity	Create new Data Shipper entity.
site-inventory	/dataShippers/{id}	GET	-	Data Shipper entity	Retrieve a single Data Shipper entity
site-inventory	/dataShippers/{id}	PUT	Data Shipper entity with updated values	Data Shipper entity	Update an existing Data Shipper entity.

[48] https://github.com/5GEVE/OpenAPI

site-inventory	/dataShippers/{id}	DELETE	-	-	Delete an existing Data Shipper entity.
site-inventory	/dataShippers/{id}	PATCH	Fields to be updated	Data Shipper entity	Update an existing Data Shipper entity.
site-inventory	/subscriptions	GET	-	Array of LccnSubscription entities	Retrieve list of LccnSubscription entities.
site-inventory	/subscriptions	POST	LccnSubscription entity	The newly created entity	Create new LccnSubscription entity.
site-inventory	/subscriptions/{id}	GET	-	LccnSubscription entity	Retrieve a single LccnSubscription entity
site-inventory	/subscriptions/{id}	PUT	LccnSubscription entity with updated values	LccnSubscription entity	Update an existing LccnSubscription entity.
site-inventory	/subscriptions/{id}	DELETE	-	-	Delete an existing LccnSubscription entity.
site-inventory	/subscriptions/{id}	PATCH	Fields to be updated	LccnSubscription entity	Update an existing LccnSubscription entity.
site-inventory	/networks	GET	-	Array of Network entities	Retrieve list of Network entities.
site-inventory	/networks	POST	Network entity	The newly created entity	Create new Network entity.
site-inventory	/networks/{id}	GET	-	Network entity	Retrieve a single Network entity
site-inventory	/networks/{id}	PUT	Network entity with updated values	Network entity	Update an existing Network entity.
site-inventory	/networks/{id}	DELETE	-	-	Delete an existing Network entity.
site-inventory	/networks/{id}	PATCH	Fields to be updated	Network entity	Update an existing Network entity.
site-inventory	/nfvOrchestrators	GET	-	Array of NfvOrchestrator entities	Retrieve list of NfvOrchestrator entities.
site-inventory	/nfvOrchestrators	POST	NfvOrchestrator entity	The newly created entity	Create new NfvOrchestrator entity.
site-inventory	/nfvOrchestrators/{id}	GET	-	NfvOrchestrator entity	Retrieve a single NfvOrchestrator entity

site-inventory	/nfvOrchestrators/{id}	PUT	NfvOrchestrator entity with updated values	NfvOrchestrator entity	Update an existing NfvOrchestrator entity.
site-inventory	/nfvOrchestrators/{id}	DELETE	-	-	Delete an existing NfvOrchestrator entity.
site-inventory	/nfvOrchestrators/{id}	PATCH	Fields to be updated	NfvOrchestrator entity	Update an existing NfvOrchestrator entity.
site-inventory	/ranOrchestrators	GET	-	Array of RanOrchestrator entities	Retrieve list of RanOrchestrator entities.
site-inventory	/ranOrchestrators	POST	RanOrchestrator entity	The newly created entity	Create new RanOrchestrator entity.
site-inventory	/ranOrchestrators/{id}	GET	-	RanOrchestrator entity	Retrieve a single RanOrchestrator entity
site-inventory	/ranOrchestrators/{id}	PUT	RanOrchestrator entity with updated values	RanOrchestrator entity	Update an existing RanOrchestrator entity.
site-inventory	/ranOrchestrators/{id}	DELETE	-	-	Delete an existing RanOrchestrator entity.
site-inventory	/ranOrchestrators/{id}	PATCH	Fields to be updated	RanOrchestrator entity	Update an existing RanOrchestrator entity.
site-inventory	/ranZones	GET	-	Array of RanZone entities	Retrieve list of RanZone entities.
site-inventory	/ranZones	POST	RanZone entity	The newly created entity	Create new RanZone entity.
site-inventory	/ranZones/{id}	GET	-	RanZone entity	Retrieve a single RanZone entity
site-inventory	/ranZones/{id}	PUT	RanZone entity with updated values	RanZone entity	Update an existing RanZone entity.
site-inventory	/ranZones/{id}	DELETE	-	-	Delete an existing RanZone entity.
site-inventory	/ranZones/{id}	PATCH	Fields to be updated	RanZone entity	Update an existing RanZone entity.
site-inventory	/sites	GET	-	Array of Site entities	Retrieve list of Site entities.
site-inventory	/sites	POST	Site entity	The newly created entity	Create new Site entity.

site-inventory	/sites/{id}	GET	-	Site entity	Retrieve a single Site entity
site-inventory	/sites/{id}	PUT	Site entity with updated values	Site entity	Update an existing Site entity.
site-inventory	/sites/{id}	DELETE	-	-	Delete an existing Site entity.
site-inventory	/sites/{id}	PATCH	Fields to be updated	Site entity	Update an existing Site entity.
site-inventory	/vimAccounts	GET	-	Array of Vim Account entities	Retrieve list of Vim Account entities.
site-inventory	/vimAccounts	POST	Vim Account entity	The newly created entity	Create new Vim Account entity.
site-inventory	/vimAccounts/{id}	GET	-	Vim Account entity	Retrieve a single Vim Account entity
site-inventory	/vimAccounts/{id}	PUT	Vim Account entity with updated values	Vim Account entity	Update an existing Vim Account entity.
site-inventory	/vimAccounts/{id}	DELETE	-	-	Delete an existing Vim Account entity.
site-inventory	/vimAccounts/{id}	PATCH	Fields to be updated	Vim Account entity	Update an existing Vim Account entity.

3.6.3 Service description

Some components of the Interworking framework activate REST API calls within their operational workflows, these calls allow access for the aforementioned components to the IWF Repository. The API access is preferred for a number of reasons with respect to a direct access to a Database Management system. Indeed, the API performs some validation checks on data entities when they are created e.g., `not null` constraints, pattern matching for UUIDs and IP addresses. Furthermore, having a REST API enables to limit the client operations on the data entities (e.g., for read-only entities) and it keeps open the option to add security mechanisms in the future (at the moment, Site-Inventory interface listens on a local subnet with restricted access through a VPN server). Spring offers some well-established modules in order to support the developer in adding security features to its application.

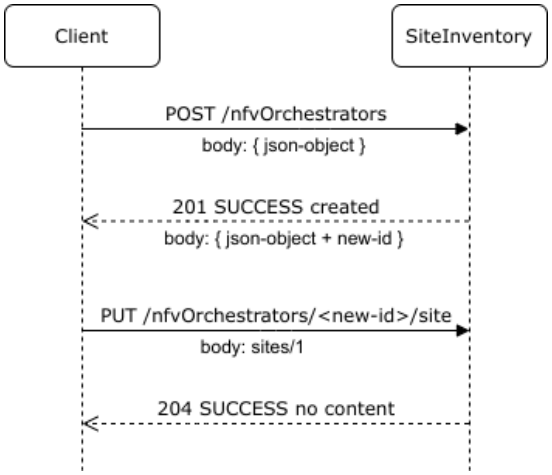

Figure 31: Example HTTP requests to create new entities in IWF Repository

The creation of new entities in the IWF Repository through the REST API must take also an additional step to create the associations. For example, let us consider the creation of a new Nfv Orchestrator entity as showed in Figure 31. We can create the new database record in the proper table by performing a POST request to /nfvOrchestrators (first message in Figure 31). The IWF Repository returns a 201-status code if this operation is successful (second message in Figure 31). Now, to associate the newly created entity with a Site entity, an additional PUT request is needed (third message in Figure 31 The request path, in our example, is '/nfvOrchestrators/{new-id}/site'. The headers must contain a Content Type header with value 'text/uri-list'. Finally, the request body must contain the path of the Site entity we want to associate: /sites/{id-of-desired-site}. If the association has been created successfully, the IWF Repository returns a 204-status code (fourth message in Figure 31).

3.7 Site adaptations

Following we describe the work we have been done in WP3 in each of the 5G EVE sites. We include only French and Spanish sites in this document as they are the sites with relevant updates. For Italian and Greek site please refer to D3.4.

3.7.1 French site

The French Site facility is designed as a star topology within a central management point located in Chatillon, Paris. Several geo distributed entities located in Rennes, Lannion, Saclay and Sophia Antipolis are surrounding Paris and connects via VPN. 5G-EVE portal treats each of those locations as a single French sub-site, where NS might be instantiated.

The Open Network Automation Platform (ONAP) solution has been selected as a main tool of management and orchestration in French Site. It is an open-source, complete solution that provides a comprehensive platform for real-time, policy-driven service orchestration and automation. In the French Site ONAP is deployed in Chatillon with five different Virtual Infrastructure Managers (VIMs) connected, where NS can be deployed. NS can be instantiated on OpenStack instances (Chatillon, Lannion, Rennes, Saclay) or OpenShift clusters (Sophia Antipolis). In order to integrate ONAP and the I/W Framework (IWL), ONAP driver in MSO-LO and Translation Component (TC) in French Site were implemented. The final implementation of MSO-LO with ONAP driver provides all required operation for the management of NS and VNF instances and notification systems. Architectural integration of IWL with French Site with ONAP is depicted in Figure 32.

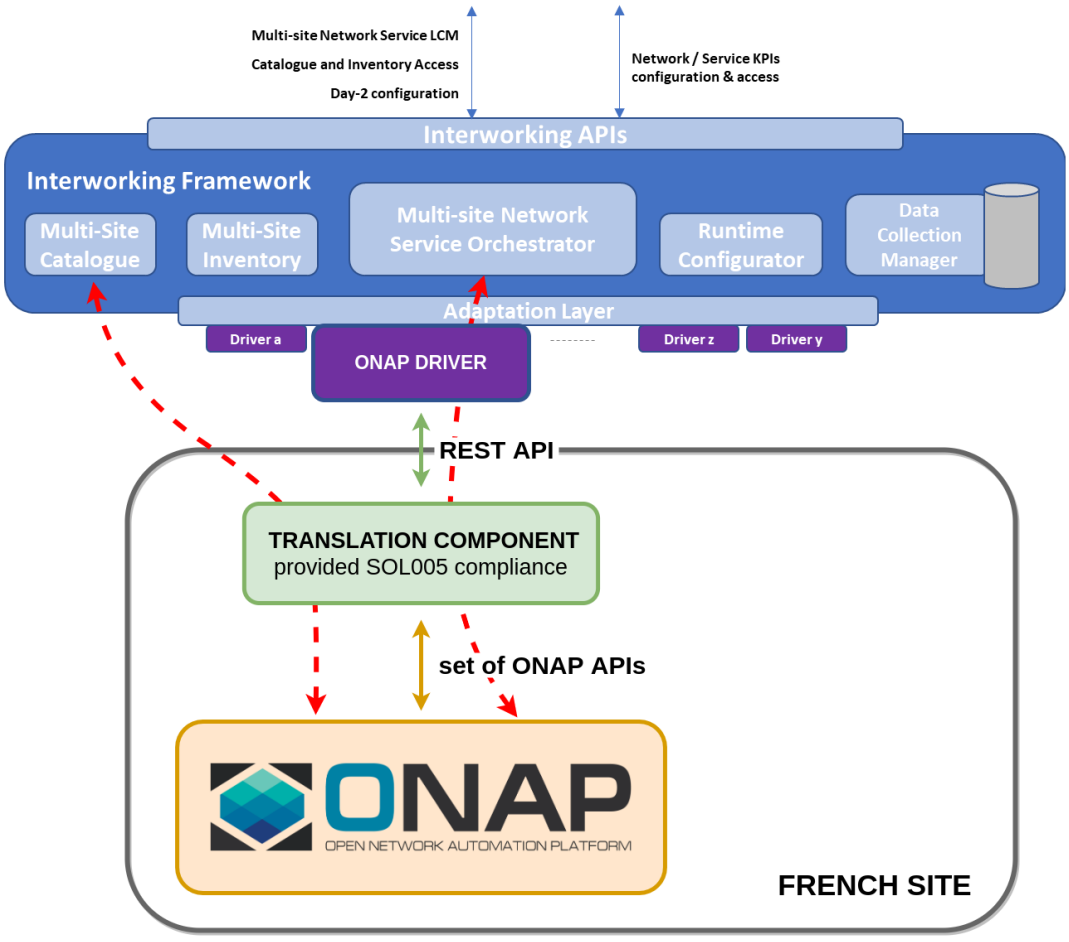

Figure 32: The overview architecture of French Site

Translation Component (TC) has been implemented in order to integrate French Site with IWL. Main responsibility of TC is to automate deployment process of network services (NS) and provide ONAP a communication compliance with ETSI NFV SOL005 standard. As far as MSNO is communicating using the SOL005 standard and ONAP uses its own custom interfaces - we introduced a component that is translating SOL005 requests and calls a set of ONAP APIs. To be more precise - when any request comes from IWL via ONAP Driver to French Sites - in the first step it is passed to TC that has defined REST APIs and may be treated as an "external interface of French Site". Depending on the request type, TC triggers proper action using onapsdk [20] python package to interact with ONAP APIs to manage the life cycle of NS instances.

TC provided the following basic functionalities required by 5G-EVE IWL:

- create, instantiate, terminate and delete network service instances;
- retrieve a list of network services and life-cycle management operation occurrences;
- retrieve information about selected network service or life-cycle management operation occurrence resource;
- notification about life-cycle management operation occurrences of network services.

Taking into account the design of the French Site, TC has to provide a functionality to select the target French sub-site for NS instances. To achieve that, all sub-sites in the French node have to be configured in TC. Each of them is treated as a single VIM with a unique identifier (vimShortId). That id has to be in line with the SiteId in the IWF Repository. The information about the selected localisation for NS instance within the experiment is passed from portal to IWL. Next, the ID of the target sub-site is passed to TC by the ONAP Driver in the body of "create ns instance Id" request. Next, NS instance is deployed in a proper sub-site.

TC provides also an integration of ONAP and MS-Catalogue. The main goal is to synchronize services' descriptors managed by ONAP with IWL Catalog. TC implements dedicated API to retrieve service specification from French Site. It allows MS-Catalogue to access ONAP and retrieve selected service specifications as a tosca csar archive. MS-Catalog implements a dedicated tool to translate ONAP service model into the format required by IWL. It allows to receive and store specific information about services like: topology, resource requirements, services management interfaces. More detailed information regarding this custom tool can be found in MS-Catalogue description. Information retrieved in a process of ONAP-IWL catalog synchronization can be utilized by other components, like portal - to present detailed information about use cases in French Site.

Table 9 describes in detail the API methods offered by Translation Component provided SOL005 compliance for ONAP.

Table 9: API provided by Translation Component as a "external API of French Site".

Service	Path	Method	Input	Output	Description
Translation Component	/instances	GET	-	Array of NS instances	Retrieve a list of NS instances
Translation Component	/instances/ {nsInstanceId}	GET	-	NS instance	Read an individual NS instance resource.
Translation Component	/create	POST	nsdId, nsName, nsDescription vimShortId	NS identifier	Creates and returns a Network Service identifier (nsId) in a Nfvo

Translation Component	/instantiate/ {nsInstanceId}	POST	-	-	Instantiate a NS instance.
Translation Component	/terminate/ {nsInstanceId}	POST	-	-	Terminate a NS instance
Translation Component	/delete/ {nsInstanceId}	DELETE	-	-	Delete a NS instance
Translation Component	/ns_lcm_op_occs	GET	-	Array of LCM	Retrieve a list of NS LCM operation occurrences.
Translation Component	/ns_lcm_op_occs/ {nsLcmOpOccId}	GET	-	LCM Info	Retrieve an individual NS LCM operation occurrence resource.
Translation Component	/ns_lcm_op_occs/ ns_id/ {nsInstanceId}	GET	-	Array of LCM for NS instance	Retrieve a list of NS LCM operation occurrences for NS instance
Translation Component	/service_specification	GET	-	Array of NSD	Retrieve a list of NSD
Translation Component	/service_specification/{nsdId}	GET	-	csar archive for NSD	Download csar archive with service tosca model
Translation Component	/vims	GET	-	Array of configured vims	Retrieve a list of configured vims in TC

3.7.2 Greek site

No new content for Greek site, please refer to [16].

3.7.3 Italian site

No new content for Italian site, please refer to [16].

3.7.4 Spanish site

In order to carry out the experimentation planned to be performed at the Spanish site, here we present an NFV system mainly constituted by a software MANO stack and two standalone private cloud platforms as depicted in Figure 33. The cloud platforms comprise the NFV infrastructure that enables the deployment of VNFs both at the cloud and/or at the edge sides, depending on the experimentation procedure. Related with the resource orchestration, ETSI OSM is in charge of providing the MANO implementation and, in particular, this platform utilizes OSM Release SEVEN [6]. To facilitate the installation and the deployment of the NFVO and VNFM

stack, ETSI OSM is executed as a virtual machine within the 5TONIC laboratory and in compliance with the requirements established by the OSM community.

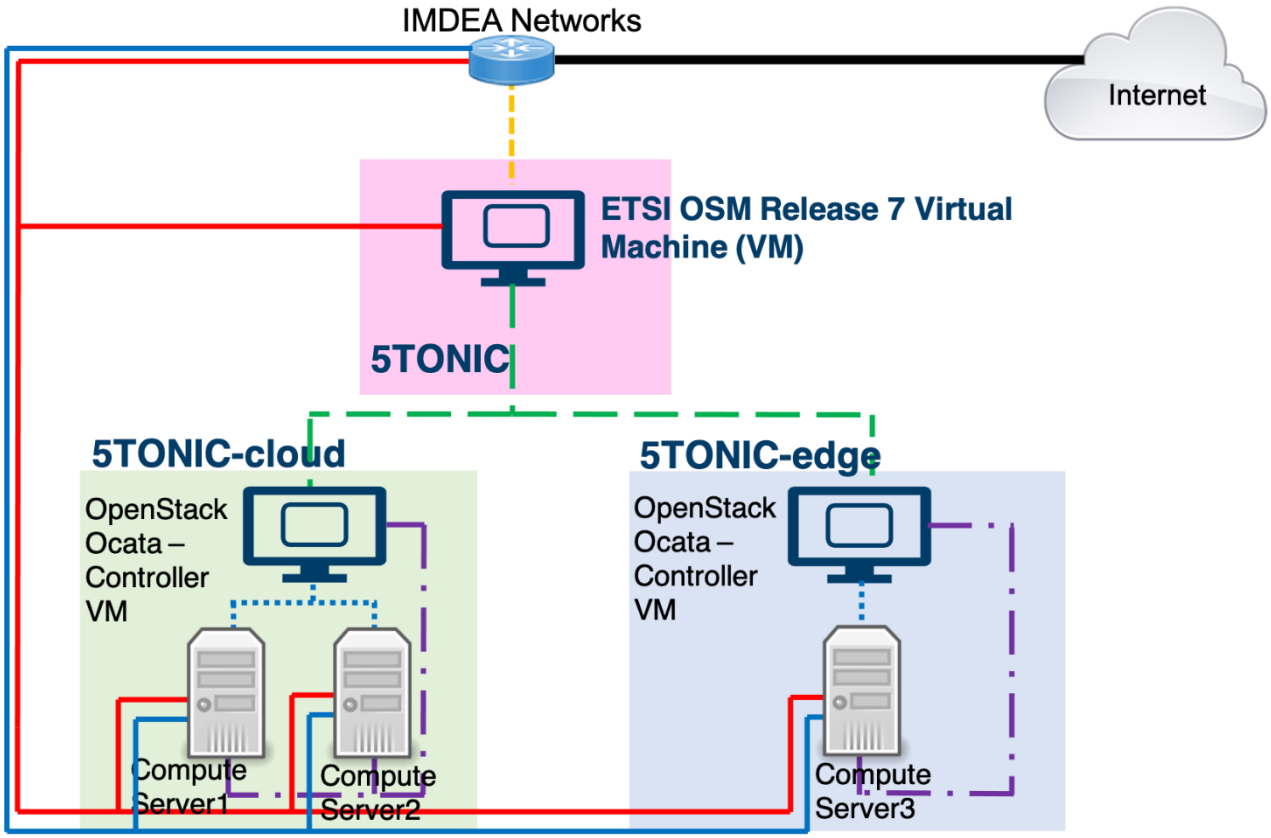

Figure 33. MANO at the Spanish site

The characteristics of the ETSI OSM, VIM and all servers in this infrastructure were already presented in D3.4, so in this document we only present the updates done since that report.

It is worth mentioning that the testbed also integrates the required networking that enable the proper operation of the complete NFV system, supporting the orchestration and deployment of multi-site network services over the three cloud platforms comprising the NFVI. These networks can be summarized as follows:

- External networks, which are exposed externally to other sites, using the VPN tunnel connecting the Spanish site with the Italian one. These networks are:
    - Orchestrator-to-VNF networks (i.e., the red solid line in Figure 33): the objective in this case is to allow the ETSI OSM platform to control and monitor the VNFs lifecycle (i.e., get the VNFs state information and support the scaling operations based on this information), as well as the VNF configuration. For this reason, ETSI OSM requires IP connectivity with each of the VNFs hosted by their respective platform. This network is exposed outside the infrastructure to allow other 5G EVE components, like the Runtime Configurator, to manage all VNFs.
    - IWL network (i.e., the orange dotted line in Figure 33): this network is used to provide access to the Spanish OSM to the IWL components hosted by TIM at the Turin site.
    - User plane network (i.e., the blue solid line in Figure 33): this network is used to connect the user/data plane network of some VNFs/PNFs with the user/data plane at other sites. For security purposes, only a subset of all the range dedicated to the user plane is allowed to connect with other sites.

- Internal networks, used only internally at the Spanish site, which are not exposed to other sites. These networks are:
    - Infrastructure management networks (i.e., the blue round dot line in Figure 33): the aim of these types of networks is to allow each VIM to be capable of managing the computational resources of the cloud platform under its control. Therefore, these networks are present independently in all the parts of the NFVI testbed.
    - Orchestrator-to-VIM networks (i.e., the green long dash line in Figure 33): these networks are in charge of supporting the reservation and allocation of the necessary resources for enabling the subsequent network services deployment. In addition, these networks are in charge of the lifecycle management of the stated network services and slices. In this context, the ETSI OSM is able to interact with each of the VIMs included in the testbed and handle the designed deployments in the experimentation procedure.
    - Service-oriented networks (i.e., the purple long dash dot line in Figure 33): this latter encompasses the networks among the different VNFs comprising a network service (regardless of the platform on which they are executed) to ensure the proper functioning of the implemented service.

3.7.4.1 Network Controller

We include a new orchestrator in Spanish site, called Network Controller, which is in charge of configuring the Radio Access Network in 5Tonic lab. With this, we allow to automatically adapt the laboratory to the conditions required for an experiment. The network controller allows to:

- Select the RAN technology: 4G, 5G NSA, 5G SA;
- Select the coverage area of the experiment;
- Select the Network Slice (eMBB, URLLC, mMTC);
- Leverage the uplink/downlink throughput: LTE-5G Aggregation, Change of TDD patterns.

4 Inter-site connectivity status[49]

As reported in D3.3 [3], we selected a star architecture for connecting the 5G EVE sites, being Turin's site the centre of the connectivity. Using this, we selected a distributed deployment for the 5G EVE Framework, with the 5G EVE Portal in 5TONIC and the IWL in Turin's site.

As we planned, we interconnect three networks:

- Orchestration and Management network, which is required for connecting 5G EVE components and for the IWL-site connectivity.
- Control plane network, for connecting 5G Core elements.
- User plane network, for connecting elements in the data path of the 5G users. We setup and test the connectivity between Spanish and Greek sites and we use the setup in the Gaming use case demonstration.

[49] This section is almost equal to section 4 reported in D3.4. We included the full text for maintaining the clarity, just adding the update of the user plane setup.

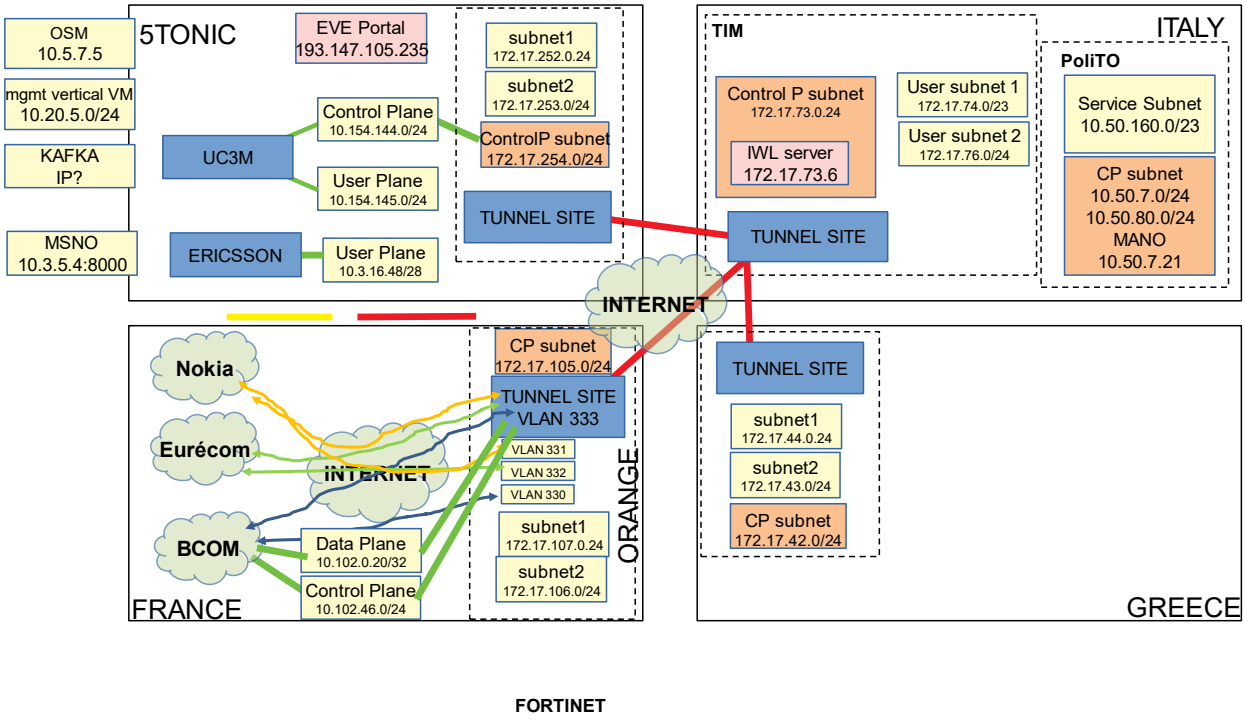

Figure 34: 5G EVE site integration

One issue we faced is the IP addressing overlapping among the 5G EVE sites. Since this is not an issue easy to solve, as it involves parts of the networks that are not related with 5G EVE project, we decided to use an IP translation system, which allows us to define common subnets for all the 5G EVE sites. We describe in Figure 34 the integration of the 5G EVE lab with TIM's lab.

The performance measurements of the inter-site connectivity are described in the Deliverable 3.7, which will be published at the same time of this deliverable.

5 Updated roadmap

In Table 10 we show the summary of the features supported by IWL, according to the plan exposed in D 3.2.

Table 10: 5G EVE Interworking Framework Roadmap

Key Features	Category	Brief Description	D3.3 (M16)	Drop 1 (M18)	Drop 2 (M22)	D3.4 (M24)	Final Delivery
Single-Site scenarios							
Orchestration Plane Interworking	Connectivity	Low Bandwidth performance but secure connectivity among sites for orchestration traffic	Italian and French sites	Full provided	Full provided	Full provided	Full provided
Single-site Experiment Monitoring Support	Monitoring	Capability of translating the monitoring requirements defined by experimenters (based on selected KPIs) to the requested site. Sites will typically have different local monitoring tools and mechanisms.	-	Support of basic UC (one per site)	Full provided	Full provided	Full provided
Single-site Applications Deployment Support	Operation	Capability to deploy the required VNFs, hosted in the 5G EVE Catalogue, at the requested site. Sites will typically have different local orchestrators.	-	Support of basic UC (one per site)	Full provided	Full provided	Full provided
Single-site Network Automation Support	Operation	Capability to deploy the required connectivity services (first phase) and slices (second phase) to the requested site. Sites will typically have available different local controllers and network infrastructure.	-	Support of basic UC (one per site)	Full provided	Full provided	Full provided
Multi-Site scenarios							
Control Plane Interworking	Connectivity	Low Bandwidth performance but secure connectivity among sites for control traffic	-	-	Demonstrated with one selected UC	Full provided	Full provided
Data Plane Interworking	Connectivity	Secure connectivity among sites for user traffic. Low bandwidth performance experiments will employ best effort connectivity. High bandwidth performance experiments will employ a parallel high bandwidth low latency network, which will be available at least between two sites.	-	-	-	Depends on final UC requirements	Full provided: demonstrated between Spanish and Greek site

Multi-Site Experiment Monitoring support	Monitoring	Capability of translating the monitoring requirements defined by experimenters (based on selected KPIs) to the sites taking part in the same experiment. Sites will typically have different local monitoring tools and mechanisms.	-	-	Selected UC working in multi-site deployment	Full provided	Full provided
Multi-Site E2E Orchestration Support	Operation	Capability to deploy the required slices, and VNFs hosted in the 5G EVE Catalogue on top of them, to the sites taking part in the same experiment. Sites will typically have different local orchestrators, controllers and network infrastructure.	-	-	Selected UC working in multi-site deployment	Full provided	Full provided
Multi-Site slicing							
Vertical Slicing	Slicing	End-to-end slice that satisfies the requirement of the Vertical	-	Selected UC	Full provided	Full provided	Full provided
Multi-Site Slicing	Slicing	Vertical Slice that spans across multiple sites	-	-	Selected UC	Full provided	Full provided

6 Pending automated tests

In this Section, the pending tests to be reported from deliverable D3.7 are included. For a better readability of this chapter, only the pending tests will be included in this Section, omitting the finished tests that have been already reported in D3.7.

The summary of the previous test cases execution, reporting the number of pending tests for each test suite, can be checked in Table 11.

Table 11: Summary of test cases execution

Test	Test Suites	Test Summary		
		Number of Passed Tests	Number of Failed Tests	Number of pending Tests
Component Tests				
Multi-Site Catalogue Test Suites	**Multi-Site Catalogue – NSD Management**	6		9
	Multi-Site Catalogue – VNF Management	2	-	4
	Multi-Site Catalogue – PNFD Management	-	-	6
Multi-Site Inventory Test Suites	**Multi-Site Inventory – Write Operation**	3	-	6
	Multi-Site Inventory – Query Operation	2	-	1
Multi-Site Network Orchestrator Test Suites	**Multi-Site Network Orchestrator**	3	-	2
Data Collection Manager Test Suites	**Data Collection Manager Kafka**	5	-	-
	Data Collection Manager Kafka – ELK Interconnection – NBI	1	-	-
	Data Collection Manager Kafka – Filebeat Interconnection – SBI	2	-	-
Runtime Configurator Test Suites	**Runtime Configurator**	3	-	-
Adaptation Layer Test Suites	**MSO-LO – Configuration MSO-LO-Turin**	3	-	-
	MSO-LO – Configuration MSO-LO-RF	4	-	-
	MSO-LO – Configuration MSO-LO-Ever	1	-	2

Integration Tests				
NSO-Catalogue integration	**Multi-Site Network Orchestrator-Catalogue**	3	-	1
System Tests				
Site-facilities' interconnection Test Suites	**IWL – French Site**	2	-	-
	IWL – Spanish Site	2	-	-
	IWL – Italian Site	2	-	-
	IWL – Greek Site	2	-	-

6.1 Reporting format

The results of the execution of the test cases will be reported in two different formats, following the same instructions than the ones provided in D3.7:

- Using **tables** which will follow the specific format shown in Table 12 and including the following fields:
  - Name of the test component and test suite: lists the name of the component and title of the test suites;
  - Test title/<Test ID>: each test suite consists of a set of tests specified by a title and an ID;
  - Test purpose: describes the aim of the test;
  - Test tools: tools used for executing the tests (Robot Framework is commonly used as an automated testing tool);
  - Result: outlines the test results;
  - Comments/Details: any further comments or details apart from what is described in the Result field.
- The test cases executions are uploaded on a specific Github repository[50]. In this repository, there are three main folders, related to the three main test categories covered in this testing process (i.e. Component, Integration and System tests). Each of them has different sub-folders containing each entity involved.

Table 12: Table format to report the execution of test cases.

Name of the Component and Test Suite	
Test 1 title	<Test ID>
Test purpose	Summary of what is included in D3.6 [4]. If there are some modifications, include them here.
Test tools	Tools used for executing the test.
Result	Passed/Failed/[Other].
Comments	Comments related to the test that are interesting to be shared.
Details	Specific details that should be mentioned.
Test N title	<Test ID>
Test purpose	Summary of what is included in D3.6 [4]. If there are some modifications, include them here.
Test tools	Tools used for executing the test.

[50] The repository can be found here: https://github.com/5GEVE/D3.7-Test-Suites-Results (tag d3.5).

Result	Passed/Failed/[Other].
Comments	Comments related to the test that are interesting to be shared.
Details	Specific details that should be mentioned.

6.2 Interworking Framework Component Tests

6.2.1 Multi-Site Catalogue pending tests

This section aims at reporting the Multi-Site Catalogue tests pending from what was reported in D3.7. In particular, not all of the features for NSD, VNFD and PNFD management offered by the Multi-Site Catalogue were covered at that time by individual component tests. More precisely, with reference to the test plan defined in D3.6, only a subset of the functional tests defined was developed and run. The following sub-sections report the pending tests executed, in terms of test configuration and scenario, test results and a brief final result analysis.

6.2.1.1 Test configuration, scenario and execution

The Multi-Site Catalogue pending tests have been developed and executed as automated validation tests using Robot Framework [19]. For the execution phase, two main relevant test environments have been used: the Nextworks lab, where all of the Multi-Site Catalogue developments and early validations have been performed, and the Interworking Framework setup in Turin, where the actual production IWL is deployed.

Three main test configurations have been used to run the Multi-Site Catalogue pending tests, as shown in Figure 35. The component tests related to single-site scenarios against a local site orchestrator based on OSM have been executed with the setup depicted in Figure 35a, while those against a local site orchestrator based on ONAP have been executed with the setup depicted in Figure 35b. On the other hand, the component tests related to multi-site scenarios have been performed with the configuration of Figure 35c, with two local site orchestrators based on OSM. In all the three test configurations, Robot Framework acts as test system, thus issuing the specific requests to the component under test (i.e., the Multi-Site Catalogue) and validating the responses received, when needed also interacting with local site orchestrators for additional triggers or checks.

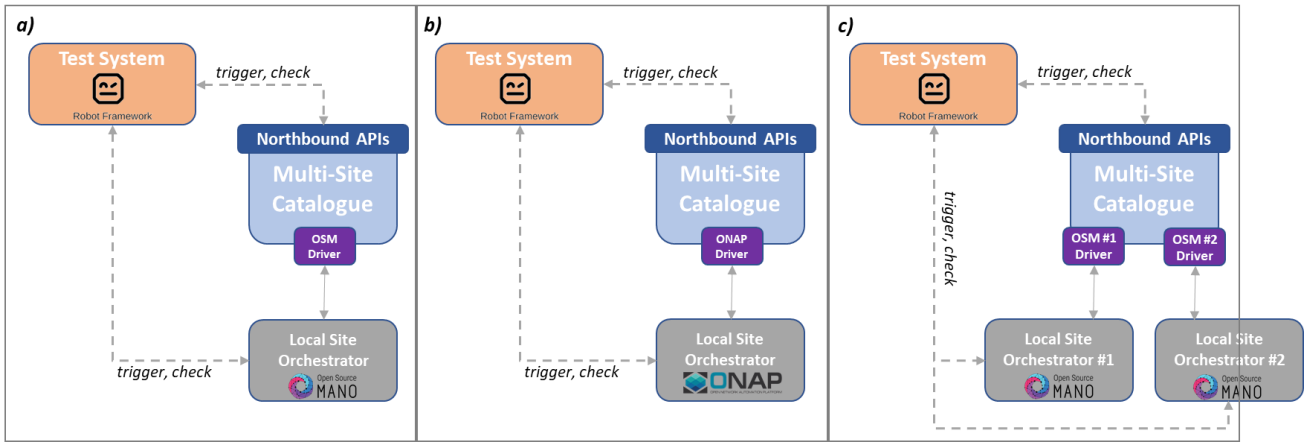

Figure 35: Multi-Site Catalogue pending tests configuration.

6.2.1.2 Test Results

Table 13, Table 14 and Table 15 provide a summary of the pending Multi-Site Catalogue test results following the template described in section 6.1 and the test plan reported in D3.6. With this new set of tests developed and executed, the Multi-Site Catalogue has been validated against both ETSI OSM and ONAP local site orchestrators, thus covering all of the possible options for the 5G EVE site facilities. Moreover, all of the Multi-Site Catalogue features have been tested at least once against ETSI OSM. Indeed, as it was agreed at project

level to keep the interaction with ONAP local site orchestrators (at the Multi-Site Catalogue level) at the bottom-up direction only (i.e. the onboard of NSDs from the 5G EVE Portal to the Multi-Site Catalogue and then to ONAP driver is not supported), some of the pending tests related to the interaction with ONAP actually became not applicable.

The tables below provide details for the automated tests executed with Robot Framework [19] as test system, thus as test trigger and validation tool against the expected behaviour for each test. The Robot Framework source code, together with the outputs generated when executing the tests are available on the project repository[51].

Table 13: Multi-Site Catalogue - NSD Management test results

Multi-Site Catalogue – NSD Management	
Test 2 - NSD Update to single site - OSM	Nsd-update-single-site-to-osm
Test purpose	This test aims at verifying that an existing NSD modelling a vertical experiment can be modified in the Multi-Site Catalogue from its NBI and updated in a specific per-site OSM orchestrator.
Test tools	Robot Framework
Result	Passed
Comments	-
Details	-
Test 5 - NSD Update from single site - OSM	Nsd-update-single-site-from-osm
Test purpose	This test aims at verifying that an existing NSD modelling a single site service (previously onboarded from a per-site OSM orchestrator) can be modified in the Multi-Site Catalogue from its SBI when it is updated in the original per-site OSM orchestrator.
Test tools	Robot Framework
Result	Passed
Comments	-
Details	-
Test 7 - NSD Onboarding to single site - ONAP	**Nsd-onboard-single-site-to-onap**
Test purpose	This test aims at verifying that an NSD modelling a vertical experiment (in TOSCA format) can be successfully onboarded in the Multi-Site Catalogue from its NBI and delivered to a specific per-site ONAP orchestrator.
Test tools	-
Result	Not Applicable
Comments	The interaction with ONAP local site orchestrators is finally possible in the bottom-up direction only
Details	-
Test 8 - NSD Update to single site - ONAP	Nsd-update-single-site-to-onap
Test purpose	This test aims at verifying that an existing NSD modelling a vertical experiment can be modified in the Multi-Site Catalogue from its NBI and updated in a specific per-site ONAP orchestrator.
Test tools	-
Result	Not Applicable

[51] https://github.com/5GEVE/D3.7-Test-Suites-Results/tree/master/component_tests/multi_site_catalogue/

Comments	The interaction with ONAP local site orchestrators is finally possible in the bottom-up direction only
Details	-
Test 9 - NSD Deletion to single site - ONAP	Nsd-delete-single-site-to-onap
Test purpose	This test aims at verifying that an existing NSD modelling a vertical experiment can be deleted in the Multi-Site Catalogue from its NBI and removed from a specific per-site ONAP orchestrator.
Test tools	-
Result	Not Applicable
Comments	The interaction with ONAP local site orchestrators is finally possible in the bottom-up direction only
Details	-
Test 10 - NSD Onboarding from single site - ONAP	Nsd-onboard-single-site-from-onap
Test purpose	This test aims at verifying that an NSD modelling a single-site service can be successfully onboarded in the Multi-Site Catalogue through the SBI from a specific per-site ONAP orchestrator.
Test tools	Robot Framework
Result	Passed
Comments	-
Details	-
Test 11 - NSD Update from single site - ONAP	Nsd-update-single-site-from-onap
Test purpose	This test aims at verifying that an existing NSD modelling a single site service (previously onboarded from a per-site ONAP orchestrator) can be modified in the Multi-Site Catalogue from its SBI when it is updated in the original per-site ONAP orchestrator.
Test tools	Robot Framework
Result	Passed
Comments	-
Details	-
Test 12 - NSD Deletion from single site - ONAP	Nsd-delete-single-site-from-onap
Test purpose	This test aims at verifying that an existing NSD modelling a per-site service (previously onboarded from a per-site ONAP orchestrator) can be deleted in the Multi-Site Catalogue from its SBI when it is removed in the original per-site ONAP orchestrator.
Test tools	Robot Framework
Result	Passed
Comments	-
Details	-
Test 14 - NSD Update to multiple sites – OSM and ONAP	Nsd-update- multiple-site-to-osm-onap
Test purpose	This test aims at verifying that an existing NSD modelling a vertical experiment can be modified in the Multi-Site Catalogue from its NBI and updated simultaneously in the specific per-site OSM and ONAP orchestrators.
Test tools	Robot Framework
Result	Passed

Comments	-
Details	-

Table 14: Multi-Site Catalogue - VNF Management test results

Multi-Site Catalogue – VNF Management	
Test 2 - VNF Update from site - OSM	vnf-update-from-osm
Test purpose	This test aims at verifying that an existing VNF can be modified in the Multi-Site Catalogue through the SBI from a specific per-site OSM orchestrator.
Test tools	Robot Framework
Result	Passed
Comments	-
Details	-
Test 4 - VNF Onboarding from site - ONAP	vnf-onboard-from-onap
Test purpose	This test aims at verifying that a VNF can be successfully onboarded in the Multi-Site Catalogue through the SBI from a specific per-site ONAP orchestrator.
Test tools	Robot Framework
Result	Passed
Comments	-
Details	-
Test 5 - VNF Update from site - ONAP	vnf-update-from-onap
Test purpose	This test aims at verifying that an existing VNF can be modified in the Multi-Site Catalogue through the SBI from a specific per-site ONAP orchestrator.
Test tools	Robot Framework
Result	Passed
Comments	-
Details	-
Test 6 - VNF Deletion from site - ONAP	vnf-delete-from-onap
Test purpose	This test aims at verifying that an existing VNF can be deleted in the Multi-Site Catalogue through its SBI from a specific per-site ONAP orchestrator.
Test tools	Robot Framework
Result	Passed
Comments	-
Details	-

Table 15: Multi-Site Catalogue – PNFD Management test results

Multi-Site Catalogue – PNFD Management	
Test 1 - PNFD Onboarding from site - OSM	pnfd-onboard-from-osm
Test purpose	This test aims at verifying that a PNFD can be successfully onboarded in the Multi-Site Catalogue through the SBI from a specific per-site OSM orchestrator.

Test tools	Robot Framework
Result	Passed
Comments	-
Details	-
Test 2 - PNFD Update from site - OSM	pnfd-update-from-osm
Test purpose	This test aims at verifying that an existing PNFD can be modified in the Multi-Site Catalogue through the SBI from a specific per-site OSM orchestrator.
Test tools	Robot Framework
Result	Passed
Comments	-
Details	-
Test 3 - PNFD Deletion from site - OSM	pnfd-delete-from-osm
Test purpose	This test aims at verifying that an existing PNFD can be deleted in the Multi-Site Catalogue through its SBI from a specific per-site OSM orchestrator.
Test tools	Robot Framework
Result	Passed
Comments	-
Details	-
Test 4 - PNFD Onboarding from site - ONAP	pnfd-onboard-from-onap
Test purpose	This test aims at verifying that a PNFD can be successfully onboarded in the Multi-Site Catalogue through the SBI from a specific per-site ONAP orchestrator.
Test tools	-
Result	Not Applicable
Comments	PNFDs not supported in ONAP and Translation Tool
Details	-
Test 5 - PNFD Update from site - ONAP	pnfd-update-from-onap
Test purpose	This test aims at verifying that an existing PNFD can be modified in the Multi-Site Catalogue through the SBI from a specific per-site ONAP orchestrator.
Test tools	-
Result	Not Applicable
Comments	PNFDs not supported in ONAP and its Translation Component
Details	-
Test 6 - PNFD Deletion from site - ONAP	pnfd-delete-from-onap
Test purpose	This test aims at verifying that an existing VNF can be deleted in the Multi-Site Catalogue through its SBI from a specific per-site ONAP orchestrator.
Test tools	-
Result	Not Applicable
Comments	PNFDs not supported in ONAP and its Translation Component
Details	-

6.2.1.3 Results Analysis

As described in deliverable D3.6 [4], the Multi-Site Catalogue test plan was composed by a total of 27 test cases (15 for NSD Management, 6 for VNF Management and 6 for PNF Management). Out of these 27 Multi-Site Catalogue planned tests, 8 were already successfully performed and reported in deliverable D3.7. The 19 MSC pending tests are listed in Table 11, and mostly refer to the interaction with ONAP based local site orchestrators, as well as to dynamic updates of NSDs and VNFDs, and PNFD management. Out of these 19 tests, 13 have been successfully executed as part of the work reported in this deliverable, while the remaining 6 became not applicable, as it was agreed at WP3 and project level to not support (and therefore implement) the related functionalities that were originally designed and considered in deliverable D3.2 [REF]. In particular, two main categories of not applicable tests emerged: i) tests for interactions with ONAP based local site orchestrators for top-down operations (i.e., in the direction 5G EVE portal towards the local site catalogues, through the Multi-Site Catalogue), ii) tests for PNFD management against ONAP based local site orchestrators. For the first category, the ONAP used in the French site (i.e., the only site using ONAP as local orchestrator) and the related Translation Tool were agreed to support the onboarding (and update) of NSDs in the bottom-up direction only (i.e., from ONAP to the Multi-Site Catalogue). For the second category, PNFs and PNFDs were agreed to not be supported by the ONAP used in the French site. It is worth to mention that the non-applicability of such tests (and related features) had no impact in the support of all of the 5G EVE and ICT-19 vertical experiments at the Multi-Site Catalogue and IWL framework as a whole.

In practice, all the applicable and required Multi-Site Catalogue features have been implemented and successfully tested. This has enabled a full use of the Multi-Site Catalogue within the IWL first, and the whole 5G EVE platform as well, as part of all of the multi-site vertical experiments carried out in the project. Indeed, the Multi-Site Catalogue has been successfully tested and validated against all of the 5G EVE site facilities, supporting both single-site and multi-site experiments.

In summary, as reported in Table 16, among the applicable tests the results show an overall 100% execution rate and 100% success rate. If we consider also the not applicable tests (that however had no impact in the support of 5G EVE and ICT-19 vertical experiments) the results show an overall 78% execution rate with a 100% success rate.

Table 16: Summary of Multi-Site Catalogue test cases execution

Test	Test Suites	Test Summary		
		Number of Passed Tests	Number of Failed Tests	Number of Not Applicable Tests
Component Tests				
Multi-Site Catalogue Test Suites	**Multi-Site Catalogue – NSD Management**	12	0	3
	Multi-Site Catalogue – VNF Management	6	0	0
	Multi-Site Catalogue – PNFD Management	3	0	3

6.2.2 Multi-Site Inventory pending tests

Following we report the pending tests on the Multi-Site Inventory, mainly related to Multi-Site use cases.

The following tests were defined but finally not executed because the features (scaling, update) described in the test case are not finally implemented:

- **Test 2 - NSI scaling at single site**
- **Test 3 - NSI update at single site**

- **Test 6 - Composite NSI scaling**
- **Test 7 - Composite NSI update**

Table 17: Multi-Site Inventory - Write Operation test results

Multi-Site Inventory – Write Operation	
Test 2 - NSI scaling at single site	write-nsi-lifecycle-single-site-scale
Test purpose	This test aims at verifying that after the Multi-Site NSO scales a NS in a single site, it correctly updates the appropriate existing entry at the Multi-Site Inventory.
Test tools	Robot Framework [19]
Result	-
Comments	Discarded as feature is not required
Details	-
Test 3 - NSI update at single site	write-nsi-lifecycle-single-site-update
Test purpose	This test aims at verifying that after the Multi-Site NSO updates a NS in a single site, it correctly updates the appropriate existing entry at the Multi-Site Inventory.
Test tools	Robot Framework [19]
Result	-
Comments	Discarded as feature is not required
Details	-
Test 5 - Composite NSI instantiation	write-nsi-lifecycle-composite-instance
Test purpose	This test aims at verifying that after the Multi-Site NSO instantiates a composite NS in at least two sites, it correctly updates the Multi-Site Inventory, creating a new entry. The entry must include all the information related to the NS, in particular, its VNFs/PNFs and Virtual Links, and at which site they are deployed.
Test tools	Robot Framework [19]
Result	Passed
Comments	Evidence included in D3.7 repository
Details	-
Test 6 - Composite NSI scaling	write-nsi-lifecycle-composite-scale
Test purpose	This test aims at verifying that after the Multi-Site NSO scales a composite NS, it correctly updates the appropriate existing entry at the Multi-Site Inventory.
Test tools	Robot Framework [19]
Result	-
Comments	Discarded as feature is not required
Details	-
Test 7 - Composite NSI update	write-nsi-lifecycle-composite-update
Test purpose	This test aims at verifying that after the Multi-Site NSO updates a composite NS, it correctly updates the appropriate existing entry at the Multi-Site Inventory.
Test tools	Robot Framework [19]
Result	-

Comments	Discarded as feature is not required
Details	-
Test 8 - Composite NSI termination	write-nsi-lifecycle-composite-delete
Test purpose	This test aims at verifying that after the Multi-Site NSO terminates a composite NS, it correctly deletes the appropriate existing entry at the Multi-Site Inventory.
Test tools	Robot Framework [19]
Result	Passed
Comments	Evidence included in D3.7 repository
Details	-

Table 18: Multi-Site Inventory - Query Operation test results

Multi-Site Inventory – Query Operation	
Test 3 - Non-existing NSI status	query-non-existing
Test purpose	This test aims at verifying that appropriate error handling is done in case the Multi-Site NSO requests information about a non-existing NSI index at the Multi-Site Inventory.
Test tools	Robot Framework [19]
Result	Passed
Comments	Evidence included in D3.7 repository
Details	-

With these results we implement Unit Tests that verify the Multi Site implementation of the MSNO.

6.2.3 Multi-Site Network Orchestrator pending tests [ERI-ES]

Following the same approach of Multi-Site Inventory, we discarded the following test cases as the features referred in the test cases (scaling, update, notifications) are not finally developed.

Table 19: Multi-Site Network Orchestrator test results

Multi-Site Network Orchestrator	
Test 4 – NS Lifecycle management operations	ms-nso-ns-lcm-op
Test purpose	The purpose of this test is to validate the LCM operations of the NS LCM.
Test tools	Robot Framework [19]
Result	Discarded
Comments	Discarded as feature is not required
Details	-
Test 5 – Network Service Lifecycle Management notifications	ms-nso-ns-lcm-not
Test purpose	The purpose of this test is to validate the MSNO notifications.
Test tools	Robot Framework [19]
Result	Discarded
Comments	Discarded as feature is not required
Details	-

The case of notifications is relevant, as we still believe that for a large-scaling solution we cannot relay in the pulling of the information and we would require an end to end notification system. But it is true that in the scope of this project, the pulling approach is more than enough, and we decided to focus our efforts on other issues (like for example, the network translations issues) needed for the success of the project

6.2.4 Adaptation Layer - EVER driver pending tests

The two tests concerning the integration with the IWL EVER driver using the robot framework and integration with EVER site and its infrastructure. Both tests are passed, and reports are submitted in github. Just to note that Test 3 is automatically done when robot framework is used for Test 2.

Table 20: Test summary for EVER driver Test suite

MSO-LO – Configuration MSO-LO-Ever	
Test 2 – EVER driver Integration	ever-driver-integration
Test purpose	Assert that the request bodies sent to EVER NBI interface are correct. Assert that responses from EVER are handled correctly. Assert that responses returned by Mso-Lo API are correct.
Test tools	Automatically via Robot Framework [19]
Result	Passed
Comments	Need RAN service slice extension in MSO-LO interface
Details	Planned to be executed in Q3 2020 and reported in D3.5
Test 3 – EVER site integration	ever-site-integration
Test purpose	Assert that IWL request bodies sent to EVER NBI interface are correct. Assert that responses from EVER are handled correctly by IWL component. Assert that radio and transport infrastructure in Italian site are correctly configured.
Test tools	Automatically (via MSNO)
Result	Passed
Comments	Need RAN service slice extension in MSO-LO interface
Details	Planned to be executed in Q3 2020 and reported in D3.5

The tests report good results in terms of delay (it takes about 2 minutes) to perform a complete lifecycle management for a RAN network slice (i.e., a complete setup and teardown of the service). The setup of EVER site integration is also reported, which is very fast (about few seconds). In addition, we tried the test multiple times, and we frequently used the driver without any problem occurring (so the software is quite robust).

6.3 Integration Tests between Interworking Framework components

6.3.1 NSO-Catalogue pending tests

Table 21: Multi-Site Network Orchestrator-Catalogue test results

Multi-Site Network Orchestrator-Catalogue	
Test 4 – Network Service Lifecycle Management notifications	ms-nso-ns-lcm-not
Test purpose	The purpose of this test is to validate the MSNO notifications.
Test tools	Robot Framework [19]

Result	Not executed
Comments	Discarded as notifications are not in the final scope
Details	-

7 Conclusions

We present in this document the result of the work of WP3 in the whole project.

In our opinion, we demonstrate that the IWL designed, developed and tested in this WP solves the initial problem of integrating multiple sites with heterogeneous technology, allowing to the experiments to work with a unique interface that hides the complexity behind the 5G architecture.

We also demonstrate the flexibility of the IWL design, as we incorporate new components on-the-fly: The IWF repository for configuring the 5G EVE details, new RAN orchestrators that provides capabilities of adapting the sites' Radio Access Network and even an ICT-19 driver (5Growth) to incorporate a new site orchestrated with a new orchestrator (5Growth Service Orchestrator).

Acknowledgment

This project has received funding from the EU H2020 research and innovation programme under Grant Agreement No. 815074.

References

[1] 5G EVE D3.1: Interworking capability definition and gap analysis document, available at https://www.5g-eve.eu/wp-content/uploads/2019/01/5geve-d3.1-interworking-capability-gap-analysis.pdf

[2] 5G EVE D3.2: Interworking reference model, available at https://www.5g-eve.eu/wp-content/uploads/2019/09/5geve_d3.2-interworking-reference-model.pdf

[3] 5G EVE D3.3: First implementation of the interworking reference model, available at https://doi.org/10.5281/zenodo.3628179

[4] 5G EVE D3.6: Interworking test suites, available at https://doi.org/10.5281/zenodo.3628189

[5] ETSI GS NFV-SOL 005 V2.6.1: Network Functions Virtualisation (NFV) Release 2; Protocols and Data Models; RESTful protocols specification for the Os-Ma-nfvo Reference Point, available at https://www.etsi.org/deliver/etsi_gs/NFV-SOL/001_099/005/02.06.01_60/gs_nfv-sol005v020601p.pdf

[6] OSM, "Open Source MANO: ETSI-hosted project to develop MANO software stack aligned with ETSI NFV", Accessed on: Jun. 1, 2020.[Online]. Available: https://osm.etsi.org/

[7] 5G EVE D.2.6: Participating vertical industries planning. Available at https://doi.org/10.5281/zenodo.3628261

[8] ETSI GS NFV-SOL 001 V2.6.1: Network Functions Virtualisation (NFV) Release 2; Protocols and Data Models; NFV descriptors based on TOSCA specification. Available at https://www.etsi.org/deliver/etsi_gs/NFV-SOL/001_099/001/02.06.01_60/gs_NFV-SOL001v020601p.pdf

[9] 5G EVE D4.1: Experimentation tools and VNF repository. Available at https://doi.org/10.5281/zenodo.3628201

[10] ETSI GS NFV-SOL 004 v2.4.1: Network Functions Virtualisation (NFV) Release 2; Protocols and Data Models; VNF Package specification .Available at https://www.etsi.org/deliver/etsi_gs/NFV-SOL/001_099/004/02.05.01_60/gs_nfv-sol004v020501p.pdf

[11] 3GPP TS28.530, V15.1.0, Telecommunication management; Management of 5G networks and network slicing; Concepts, use cases and requirements, 2018

[12] 3GPP TS28.540, V15.1.0 Management and orchestration; 5G Network Resource Model (NRM); Stage 1

[13] E. Gamma, "Design patterns: elements of reusable object-oriented software," Pearson Education India, 1995.

[14] Ramon Perez, Jaime Garcia-Reinoso, Aitor Zabala, Pablo Serrano, Albert Banchs, “A Monitoring Framework for Multi-Site 5G Platforms”, 2020 European Conference on Networks and Communications (EuCNC): Operational \& Experimental Insights (OPE), pp. 52-56, 2020.

[15] ONAP TOSCA model: available at https://wiki.onap.org/display/DW/4.+ONAP+Internal+TOSCA+Modeling+and+Data+Model

[16] 5G EVE D3.4: Second implementation of the interworking reference model, available at https://doi.org/10.5281/zenodo.3946323

[17] H2020 5Growth, https://5growth.eu/

[18] H2020 5Growth Deliverable D3.3 “First version of software implementation for the platform”, https://5growth.eu/wp-content/uploads/2019/06/D3.3-First_version_of_software_implementation.pdf

[19] “Robot Framework” – Available at https://robotframework.org/

[20] Python ONAP SDK, sdk to use onap programmatically with python code,” https://gitlab.com/Orange-OpenSource/lfn/onap/python-onapsdk

[21] ETSI NFV Specifications: ETSI - Standards for NFV - Network Functions Virtualisation | NFV Solutions (last visited 2021-05-24)